

International Journal on Advances in Software



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

ISSN: 1942-2628

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

contact: petre@iaria.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. 4, no.3 & 4, year 2011, <http://www.iariajournals.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. 4, no. 3 & 4, year 2011,<start page>:<end page>, <http://www.iariajournals.org/software/>*

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

Sponsored by IARIA

www.iaria.org

Copyright © 2011 IARIA

Editor-in-Chief

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

Editorial Advisory Board

Meikel Poess, Oracle, USA

Hermann Kaindl, TU-Wien, Austria

Herwig Mannaert, University of Antwerp, Belgium

Editorial Board

 **Software Engineering**

- Marc Aiguier, Ecole Centrale Paris, France
- Sven Apel, University of Passau, Germany
- Kenneth Boness, University of Reading, UK
- Hongyu Pei Breivold, ABB Corporate Research, Sweden
- Georg Buchgeher, SCCH, Austria
- Dumitru Dan Burdescu, University of Craiova, Romania
- Angelo Gargantini, Universita di Bergamo, Italy
- Holger Giese, Hasso-Plattner-Institut-Potsdam, Germany
- Jon G. Hall, The Open University - Milton Keynes, UK
- Herman Hartmann, NXP Semiconductors- Eindhoven, The Netherlands
- Hermann Kaindl, TU-Wien, Austria
- Markus Kirchberg, Institute for Infocomm Research, A*STAR, Singapore
- Herwig Mannaert, University of Antwerp, Belgium
- Roy Oberhauser, Aalen University, Germany
- Flavio Oquendo, European University of Brittany - UBS/VALORIA, France
- Eric Pardede, La Trobe University, Australia
- Aljosa Pasic, ATOS Research/Spain, NESSI/Europe
- Robert J. Pooley, Heriot-Watt University, UK
- Vladimir Stantchev, Berlin Institute of Technology, Germany
- Osamu Takaki, Center for Service Research (CfSR)/National Institute of Advanced Industrial Science and Technology (AIST), Japan
- Michal Zemlicka, Charles University, Czech Republic

 **Advanced Information Processing Technologies**

- Mirela Danubianu, "Stefan cel Mare" University of Suceava, Romania
- Michael Grottke, University of Erlangen-Nuremberg, Germany
- Josef Noll, UiO/UNIK, Sweden
- Olga Ormandjieva, Concordia University-Montreal, Canada

- Constantin Paleologu, University ‘Politehnica’ of Bucharest, Romania
- Liviu Panait, Google Inc., USA
- Kenji Saito, Keio University, Japan
- Ashok Sharma, Satyam Computer Services Ltd – Hyderabad, India
- Marcin Solarski, IBM-Software Labs, Germany

Advanced Computing

- Matthieu Geist, Supelec / ArcelorMittal, France
- Jameleddine Hassine, Cisco Systems, Inc., Canada
- Sascha Opletal, Universitat Stuttgart, Germany
- Flavio Oquendo, European University of Brittany - UBS/VALORIA, France
- Meikel Poess, Oracle, USA
- Kurt Rohloff, BBN Technologies, USA
- Said Tazi, LAAS-CNRS, Universite de Toulouse / Universite Toulouse1, France
- Simon Tsang, Telcordia Technologies, Inc. - Piscataway, USA

Geographic Information Systems

- Christophe Claramunt, Naval Academy Research Institute, France
- Dumitru Roman, Semantic Technology Institute Innsbruck, Austria
- Emmanuel Stefanakis, Harokopio University, Greece

Databases and Data

- Peter Baumann, Jacobs University Bremen / Rasdaman GmbH Bremen, Germany
- Qiming Chen, HP Labs – Palo Alto, USA
- Ela Hunt, University of Strathclyde - Glasgow, UK
- Claudia Roncancio INPG / ENSIMAG - Grenoble, France

Intensive Applications

- Fernando Boronat, Integrated Management Coastal Research Institute, Spain
- Chih-Cheng Hung, Southern Polytechnic State University, USA
- Jianhua Ma, Hosei University, Japan
- Milena Radenkovic, University of Nottingham, UK
- Djamel H. Sadok, Universidade Federal de Pernambuco, Brazil
- Marius Slavescu, IBM Toronto Lab, Canada
- Cristian Ungureanu, NEC Labs America - Princeton, USA

Testing and Validation

- Michael Browne, IBM, USA
- Cecilia Metra, DEIS-ARCES-University of Bologna, Italy
- Krzysztof Rogoz, Motorola, USA
- Sergio Soares, Federal University of Pernambuco, Brazil
- Alin Stefanescu, University of Pitesti, Romania

- Massimo Tivoli, Universita degli Studi dell'Aquila, Italy

❖ Simulations

- Robert de Souza, The Logistics Institute - Asia Pacific, Singapore
- Ann Dunkin, Hewlett-Packard, USA
- Tejas R. Gandhi, Virtua Health-Marlton, USA
- Lars Moench, University of Hagen, Germany
- Michael J. North, Argonne National Laboratory, USA
- Michal Pioro, Warsaw University of Technology, Poland and Lund University, Sweden
- Edward Williams, PMC-Dearborn, USA

CONTENTS

The OMISCID 2.0 Middleware: Usage and Experiments in Smart Environments	231 - 243
Rémi Barraquand, INRIA, France	
Dominique Vaufreydaz, INRIA, France	
Rémi Emonet, INRIA, France	
Amaury Negre, INRIA, France	
Patrick Reignier, INRIA, France	
On a New Method for Derivative Free Optimization	244 - 255
Lennart Frimannslund, Department of Informatics, University of Bergen, Norway	
A Systematic Review and Taxonomy of Runtime Invariance in Software Behaviour	256 - 274
Teemu Kanstrén, VTT, Finland	
Interface Contracts for WCF Services with Code Contracts	275 - 285
Bernhard Hollunder, Furtwangen University of Applied Sciences, Germany	
Answering Complex Requests with Automatic Composition of Semantic Web Services	286 - 307
Brahim Batouche, Public Research Center Henri Tudor, Luxembourg, Luxembourg	
Yannick Naudet, Public Research Center Henri Tudor, Luxembourg, Luxembourg	
Frédéric Guinand, University of LeHavre, France	
Scripting Technology for Generative Modeling	308 - 326
Christoph Schinko, Institut f. ComputerGraphik & WissensVisualisierung, TU Graz, Austria	
Martin Strobl, Institut f. ComputerGraphik & WissensVisualisierung, TU Graz, Austria	
Torsten Ullrich, Fraunhofer Austria Research GmbH, Graz, Austria	
Dieter W. Fellner, GRIS, TU Darmstadt & Fraunhofer IGD, Darmstadt, Germany	
Simulation and Test-Case Generation for PVS Specifications of Control Logics	327 - 341
Cinzia Bernardeschi, Department of Information Engineering, University of Pisa, Italy	
Luca Cassano, Department of Information Engineering, University of Pisa, Italy	
Andrea Domenici, Department of Information Engineering, University of Pisa, Italy	
Paolo Masci, School of Electronic Engineering and Computer Science, Queen Mary University of London, United Kingdom	
The CloudMIG Approach: Model-Based Migration of Software Systems to Cloud-Optimized Applications	342 - 353
Sören Frey, Software Engineering Group, University of Kiel, Germany	
Wilhelm Hasselbring, Software Engineering Group, University of Kiel, Germany	

A Formal Language for the Expression of Pattern Compositions	354 - 366
Ian Bayley, Oxford Brookes University, UK	
Hong Zhu, Oxford Brookes University, UK	
A Pattern-based Adaptation for Abstract Applications in Pervasive Environments	367 - 377
Imen Ben Lahmar, Institut Telecom; Telecom SudParis, France	
Djamel Belaïd, Institut Telecom; Telecom SudParis, France	
Hamid Mukhtar, National University of Sciences and Technology, Pakistan	
On the Quality of Relational Database Schemas in Open-source Software	378 - 388
Fabien Coelho, CRI, Mathématiques et Systèmes, MINES ParisTech, France	
Alexandre Aillois, CRI, Mathématiques et Systèmes, MINES ParisTech, France	
Samuel Pilot, CRI, Mathématiques et Systèmes, MINES ParisTech, France	
Shamil Valeev, CRI, Mathématiques et Systèmes, MINES ParisTech, France	
Using Statistical Information for Efficient Design and Evaluation of Hybrid XML Storage	389 - 400
Lena Strömbäck, SMHI, Sweden	
Valentina Ivanova, Linköping University, Sweden	
David Hall, Linköping University, Sweden	
A Proposal of a New Compression Scheme of Medium-Sparse Bitmaps	401 - 411
Andreas Schmidt, Karlsruhe Institute of Technology, Germany	
Daniel Kimmig, Karlsruhe Institute of Technology, Germany	
Mirko Beine, Karlsruhe University of Applied Sciences, Germany	
Turning Large Software Component Repositories into Small Index Files	412 - 421
Marcos Paixão, Federal University of Sergipe, Brazil	
Leila Silva, Federal University of Sergipe, Brazil	
Talles Brito, Federal University of Paraíba, Brazil	
Gledson Elias, Federal University of Paraíba, Brazil	
Superposition of Rectangles with Visibility Requirement: A Qualitative Approach	422 - 433
Takako Konishi, Kwansei Gakuin University, Japan	
Kazuko Takahashi, Kwansei Gakuin University, Japan	
Efficient Non-Sequential Access and More Ordering Choices in a Search Tree	434 - 441
Lubomir Stanchev, Indiana University - Purdue University Fort Wayne, USA	
Transactional Composition and Concurrency Control in Disconnected Computing	442 - 460
Tim Lessner, Reutlingen University of Applied Sciences, Germany	
Fritz Laux, Reutlingen University of Applied Sciences, Germany	
Thomas Connolly, University of the West of Scotland, Scotland, UK	
Malcolm Crowe, University of the West of Scotland, Scotland, UK	

Verifiable Constraints for Ambients of Persistent Objects	461 - 470
Suad Alagic, University of Southern Maine, USA	
Harika Anumula, University of Southern Maine, USA	
Akinori Yonezawa, Advanced Institute of Computational Science, Japan	
Models of 40-Year Spatial Development of Cities in the Czech Republic in a geographic information system	471 - 478
Lena Halounová, Faculty of Civil Engineering, CTU in Prague, Czech Republic	
Karel Vepřek, Faculty of Civil Engineering, CTU in Prague, Czech Republic	
Martin Řehák, Faculty of Civil Engineering, CTU in Prague, Czech Republic	
Rainbow Table Optimization for Password Recovery	479 - 488
Vrizlynn Thing, Institute for Infocomm Research, Singapore	
Hwei-Ming Ying, Institute for Infocomm Research, Singapore	
An Augmented Reality Platform for the Enhancement of Surgical Decisions in Pediatric Laparoscopy	489 - 498
Lucio Tommaso De Paolis, Department of Innovation Engineering - University of Salento, Italy	
Giovanni Aloisio, Department of Innovation Engineering - University of Salento, Italy	
Retrieval of 3D Medical Images via Their Texture Features	499 - 509
Xiaohong Gao, Middlesex University, United Kingdom	
Yu Qian, Middlesex University, United Kingdom	
Martin Loomes, Middlesex University, UK	
Richard Comley, Middlesex University, UK	
Balbir Barn, Middlesex University, UK	
Alex Chapman, Middlesex University, UK	
Janet Rix, Middlesex University, UK	
Rui Hui, General Navy Hospital, China	
Zengmin Tian, General Navy Hospital, China	
Ontology Structure, Reasoning Approach and Querying Mechanism in a Semantic-Enabled Efficient and Scalable Retrieval of Experts	510 - 520
Witold Abramowicz, Poznan University of Economics, Poland	
Elżbieta Bukowska, Poznan University of Economics, Poland	
Monika Kaczmarek, Poznan University of Economics, Poland	
Monika Starzecka, Poznan University of Economics, Poland	
Block Matching Motion Estimation with Variable Search Window Size	521 - 531
Ionut Pirnog, "Politehnica" University of Bucharest, Romania	
Claudia Cristina Oprea, "Politehnica" University of Bucharest, Romania	
Automatic Categorisation of E-Journals by Synonym Analysis of n-grams	532 - 542

Richard Hussey, University of Reading, United Kingdom

Shirley Williams, University of Reading, United Kingdom

Richard Mitchell, University of Reading, United Kingdom

The OMiSCID 2.0 Middleware: Usage and Experiments in Smart Environments

Rémi Barraquand*, Dominique Vaufreydaz^{†*}, Rémi Emonet*, Amaury Nègre*, Patrick Reignier^{‡*}

*PRIMA Team - INRIA/LIG/CNRS - 655, avenue de l'Europe - 38334 Saint Ismier Cedex

see <http://www-prima.inrialpes.fr/>

[†]Université Pierre-Mendès-France - BP 47 - 38040 Grenoble Cedex 9

[‡]Université Joseph Fourier - BP 53 - 38041 Grenoble Cedex 9

{Remi.Barraquand,Dominique.Vaufreydaz,Remi.Emonet,Amaury.Negre,Patrick.Reignier,omiscid-info}@inria.fr

Abstract—OMiSCID 2.0 is a lightweight middleware for ubiquitous computing and ambient intelligence. Its main objective is to bring Service Oriented Architectures to all developers. After reviewing related works, we demonstrate how OMiSCID 2.0, compared to other available solutions, integrates easily in classical workflows without adding any constraints on the development process. A basic overview of our middleware is given along with brief technical descriptions demonstrating its *User Friendly Application Programming Interface*. This application programming interface makes it straightforward to expose, look for or send information between software components over the network. We illustrate the usage of OMiSCID 2.0, the new version of our lightweight middleware, through case-studies that have been experienced in international research projects. Particularly, we demonstrate its advantages in both development and research projects, illustrating its radical cut down effect in development time, improving software reuse and easing redeployment notably in the context of Wizard of Oz experiments conducted in smart environments.

Keywords-Service Oriented Architecture; Ubiquitous Computing; Middleware; Wizard Of Oz; Smart Environments.

I. INTRODUCTION

Today's vision of ubiquitous computing is only half way achieved. The multiplication of low cost devices along with the miniaturization of high performance computing units technically allow the design of environment widespread by cameras, motion detectors, automatic light controls, pressure sensors or microphones. Those devices, thanks to wireless networks, can communicate together with mobile and personal equipments such as cellular phones, photo frames or even personal assistants. On the other hand, the quiet and peaceful aspect of this vision where computing units can understand each others in order to collaborate is yet a research problem.

Build upon this network of devices, ambient intelligence tries to address the problem of making devices refer to users in an appropriate way by making them aware of their activity: current task, availability, focus of attention, etc. In this context, *smart environments* or *intelligent environments* refer to environments that are spread with sensors and actuators which sense users' activities and respond according to them.

In this attempt, activity understanding remains a complex and challenging problem relying on the ability to constantly aggregate information from an ever changing medium of devices and media. A medium of information, which is in constant evolution due to the fact that media and devices, come and go, break and evolve. Mobility in this context is no longer an option. In order to guaranty the best user experience, services provided to the user should be available everywhere and at any-time. The *intelligent part* –the one that guarantees the best mapping between perceptions and actions, must, in some way, be carried along with the user in its daily activities and could, for instance, find itself embedded in a mobile phone. While the user will carry his mobile phone, the later will have to dynamically adapt to the current environment, connecting to available sensors and actuators, scanning for and exchanging with available services.

Ambient intelligence, thus, relies on a large number of different fields of expertise and brings along many challenges. Among these challenges, one that plays a central role is the handling of dynamicity in software architectures. This paper addresses the use of the OMiSCID [2] middleware that fits in between the network of devices and ambient intelligence. It aims to ease the design of agile Service Oriented Architecture (SOA) and to solve constraints of pervasive computing and intelligent environments. OMiSCID manages services in the environment, by providing cross-platform/cross-language tools for easy description, discovery and communication between software components.

In the next section, we introduce the needs for such a middleware and we present our approach, focusing on key functionalities and concepts. Benefits of using OMiSCID are shown with some *refactoring and reusability* examples. Finally, the use of OMiSCID is illustrated by the design of a Wizard of Oz experiment.

II. UBIQUITOUS COMPUTING REQUIREMENTS

The overall goal of the PRIMA research group is the elaboration of a scientific foundation for interactive environments. An interactive environment requires the capabilities of perception, action and communication. An

environment is said to be perceptive if it is capable of maintaining a model of its occupants and their activities. Such a model may include the identity of individuals, an estimation of their position, their recent trajectory, as well as recognition of the activities of individuals and groups. An environment becomes active when it is capable of action. Actions may include presentation of information. They may also include the capability to manage visual and acoustic communications, as well as the capability to transport documents and material. Controlling an environment that is perceptive and active requires a capability to interact. This capability may rely on speech recognition, gesture or object manipulation interpretation, observation of people interactions. Among those challenges is the one of developing and integrating these three capabilities.

For instance, in order to build ambient intelligence applications, many software services, developed by specialists using multiple techniques and languages, must dynamically interconnect to furnish users with the best comfort as possible. In the context of international research projects, such as DARPA or EU funded projects, the problem get even more complex as the differences among research groups involved are important. Differences include habits and historical/technical backgrounds. In order to help non software-architect researchers to interact and better collaborate, we need a simple and usable solution that addresses a common problem: how to find, to interconnect and to monitor services within the context of cross-language, cross-platform and distributed applications?

To solve this problem, one can envision many different approaches. The first one is to agree, *a priori*, about a specific programming convention, e.g., languages, platforms, technologies and so on. This solution can be adopted in small groups and must be driven by underlying technologies. This, however, has many drawbacks. For instance, it may oblige people to learn a new language or a new framework. Additionally, the agreement achieved between scientists and developer involved is subject to change, depending on the evolution of technologies but also on the evolution of the team members. An alternative scenario is to list all the software components and try to find a common way to interconnect them, no matter the platforms/languages used or any other constraints that might appear.

The alternative scenario is often approached by the implementation of a middleware, aiming to abstract lower software components. Below we list properties that such a middleware should have, at least in ubiquitous computing and ambient intelligence, our research area:

- **Attractiveness:** To be attractive, a middleware must be available in several programming languages (C++ for video/audio processing, Java for lighter processing, enterprise integration or user interfaces development, scripting language for rapid prototyping) on multiple

operating systems (Windows, Linux, MacOSX/iOS, Android).

- **Extensibility:** To be accepted, the integration of a middleware in existing programs must be timeless, costless and effortless. Adding new functionalities must be as simple as possible.
- **Networking:** Networking capabilities must support a various, always stretching, range of protocols like peer to peer connections (IP address/port) or more complex interconnection protocols using service description and discovery for instance. Ad-hoc networks should also be supported. The middleware must also support the exchange of various data types between the software components, from simple text message or formatted data structure to huge data like audio/video flows.
- **maintainability and sustainability.** Maintainability includes readability of the source code, a friendly and user oriented API, predictability of the software behaviors and monitoring of running processes over the network. Sustainability is the potential for the long-term maintenance and reuse of software components. Sustainability is strongly correlated to maintainability.

Two widely used solutions are OSGi and Web Services. They are compared in Table I. OSGi [3] is the first typical solutions to provide most of the requirements listed formerly. It permits construction of Java applications locally by recruiting components. Using iPOJO [4], it is possible to declaratively describe services and requirements using annotations for instance in order to avoid writing dedicated code. Using specific adapters, like in R-OSGi [5] it is also possible to search for non local services. Exposing an OSGi service to multiple protocols (like UPnP [6], Web Services, etc.) can be easily achieved using the ROSE (also named Chameleon) [7] ecosystem over an OSGi platform. H-OMEGA [8] proposes also an alternative using UPnP for device discovery and a centralized server for code management. Nevertheless, it is difficult to combine all the evolutions of OSGi. Additionally, even if it is possible to use JNI for C/C++ application, OSGi is dedicated to Java.

Web Services [9] are also a widely used solution for distributed applications. They permit to use web technologies in order to construct distributed applications. Services are described with the *Web Services Description Language* (WSDL) and can be discovered using *WS-Discovery*. As presented in Table I, *Web Services* are not designed to handle huge data flows. Moreover, even if there are several alternatives to WSDL, like the Business Process Execution Language for Web Services (BPEL4WS) [10] or the Web Ontology Language for Services (OWL-S) [11], they all provide service descriptions that are not easy to handle for a non specialist.

Finally, the last possible solution is to use a specialized middleware, usually dedicated to a specific task and/or environment. We can illustrate this solution by focusing on

Smart Flow II [12]. This middleware is very efficient in managing the data flow from many multimedia sources at the same time on several computers. But its force is also its weakness: it is difficult to configure and to manage other type of data.

From the previous sections, we can see that none of the reviewed solutions fulfils all the identified requirements. This assumption motivated our effort to develop the OMiSCID middleware. In the following sections, we will present the underlying concept and philosophy behind the OMiSCID middleware solution.

III. OMiSCID BASICS

OMiSCID stands for Opensource Middleware for Service Communication Inspection and Discovery [2]. It was designed to answer the problem of integration and capitalization of heterogeneous code inside smart environments. OMiSCID is distributed under a MIT-like license (free for both commercial and non-commercial applications), fully open source and available on our forge [13].

A. Concepts

The OMiSCID middleware is built around 3 main concepts: services, connectors and variables. They are detailed in the following sections.

1) *Services*: A service is a piece of software that exposes, in a transparent and light way, functionalities for a specific task. Functionalities are thus visible and available for any other services over the network without any implementation constraint. A service exports its functionalities and its state through its connectors and variables. At least, a service contains:

- *name*. This variable must represent the main function of the service. It should be human readable like *Camera*;
- *class*. This variable allows to logically organize services in categories, for instance *VideoProcessing*;
- *id*. This unique id over the network is automatically generated by OMiSCID. Services can be distinguished using this id.
- *hostname*. Computer name where the service is running;
- *owner*. *Owner* is the login which starts the service on *hostname*;
- *control port*. The *control port* is a connector used to control and manage the service.

Aggregating all these information, we obtain a service description that can be used to search and interconnect services. Services in OMiSCID are self-described, as opposed to domain specific standards like Bluetooth profiles [14] or UPnP standard device categories [6] for instance. As stated by David Svensson Fors et al. [15], standards need to be exhaustive which is not that easy, even for small applications like controlling a printer.

Granularity of what is a service is dependent of the developer's choice. Nevertheless, one must choose granularity smallest as possible in order to increase reusability and maintainability. Monolithic services are not desirable: high level services will not likely be reusable in other contexts. On the opposite, tiny services, i.e., services providing too basic functionalities, are also a very bad choice as they increase communication schema and debugging cost.

Among dozens of services we developed, we can cite four examples to illustrate our granularity choice: the light controller service which is controls the light in a room, the video service that streams data from a camera over the network, the speech activity detection service that estimates whether the sound from a microphone service contains speech, and the 3D tracker service that inputs video streams from video processing services in order to compute the 3D positions of people in a room (see Section VI-A).

2) *Connectors*: Connectors are communication ports that can be instantiated by any service to exchange data with other services. Services can have several connectors to logically separate data according to their origin. Each connector is independent from the others. It is identified by a name, a human readable description and a set of sockets where it can be reached.

Connectors can send data over TCP or UDP. In this last case, OMiSCID guarantees (re)ordering of messages. In case of message lost, each peer is notified. Connectors can send data (*input* type), receive data (*output*) or both (*input/output*).

3) *Variables*: Variables describe the service and its state. A service can expose as many variables as needed. Variables are defined by these attributes:

- *Name*. Name of the variable (254 characters max);
- *Description*. A human description of the variable;
- *Type*. Type is given as a text attribute. It can be used to parse variable value;
- *Access type*. It is possible to define *constant* variable. In case of a constant variable, value of the variable cannot be changed after starting the service. Variables can also be *read only*: modification requests coming from another service will then be automatically rejected;
- *Value*. This attribute contains the value of the variable.

Any service can register to another one to receive notifications when the value of one or several variables changes.

B. Communications

Messages are atomic elements of all communications in OMiSCID. They are sent using a connector to a specific peer or to all listening services at once. The receiver will be notified that a new message is ready when it is fully available. Each message is provided with contextual information such as the service and connector it comes from.

Table I: COMPARISON OF WIDELY USED SOLUTIONS

Description	cross-language	cross-platform	Messages	Huge data flows	Service discovery over the network
OSGi approach	No (Java)	Yes	Yes	Possible	Using <i>R-OSGi</i> for instance
Web Services	Yes	Yes	Yes	Not designed for	Using <i>WS-Discovery</i>

Even if OMiSCID is not limited to these, there are 2 kinds of workflows that are usually mixed:

- A peer to peer approach. After receiving a message from a service and processing it, a response message is sent back to it. Input/output connectors are used in this case;
- A data flow approach. After receiving a message on an input connector and processing it, a message with the result is broadcasted on another output connector in order to continue the processing chain.

Message can be sent as raw binary chunk or as text, which allows lot of flexibility for developers. Binary messages are often used to stream real time data such as video or audio. Text communication can be enhanced by using XML, YAML [16] or JSON [17] format and allows for more advanced operation and extensions (see Section IV-C).

C. Service discovery in dynamic context

Also known as service discovery, the ability to browse, find and dynamically bind running services, is one of the most important features of SOA, particularly in ubiquitous computing environments. It is not uncommon to filter services based on their current state, description or provided functionalities. Filters can be used in two different ways:

- An ask-and-wait approach asking for the list of services that match a certain criterion. This procedure will wait until at least one service match or that a timeout is reached rising an exception.
- An ask-and-listen approach notifying the application by the means of callback or listener whenever a service that matches the criteria appears or disappears.

OMiSCID provides the basic logical combination of predefined search criteria (variable value/name, connector properties, etc.). They are implemented as functor (function object). For instance, to search a *Camera* service with an output connector named *data flow* or a service *Encoding* not running on the same computer, one can write the following (C++) filter:

```
Or(And(NameIs("Camera"),
       HasConnector("data flow", AnInput)),
   And(NameIs("Encoding"),
       Not(HostIs(GetLocalHostName()))))
```

It is possible to extend filter capabilities providing more complex search primitives by implementing custom functor objects. Figure 7 and Section VII-C give clues about OMiSCID service discovery capabilities.

D. OMiSCID Gui

OMiSCID provides a simple solution to declare, to discover and to interconnect services. However, in an ecosystem spread with a multitude of services, it becomes a requirement, for both the users and the developers, to have an interface helping the visualization, the monitoring, the interactions and the control of all the services. Additionally, the debugging of a service (or a federation of services) in this *wild ecosystem* can, without appropriate tools, be a painful task. Given this facts, we developed a graphical front-end to OMiSCID: the OMiSCID Gui (see Figures 1 and 2).

OMiSCID Gui is a powerful tool built over the Netbeans platform and provides the developer with a graphical interface for multiple management tasks. It inherits many of the advantages from the Netbeans platform: portability, modularity, advanced window management, etc. OMiSCID Gui comes with light core modules and is extensible at infinite. One of the core modules is a service browser that displays all the services present within the environment as well as their connectors and variables. The service browser also provides an extensible set of contextual operations to be applied on the selected services. Default operations include for example monitoring a connector (watching or sending messages) and monitoring a variable (watching changes or sending modification requests). Among all the extensions available and easily installed using the Netbeans Plugin interface, one can find:

- A simple variable plotter that can dynamically create and display evolution of (numeric) service variables (see Figure 2);
- A family of plugins that allow the display of 2D information such as video stream or custom shapes representing for instance regions of interest of a 2D/3D tracker;
- A plugin that displays a graph of the services present in the environment along with their interconnections;
- A lot of other plugins such as real time audio stream player, 3D visualization tools, cameras controls, etc.

OMiSCID Gui comes with a public plugins repository already packed with visualization, controls, debugging plugins and can be extended by developers. All Netbeans platform plugin can also be integrated into our platform and vice versa. Its ease of use makes it a must-have tool for OMiSCID development, demonstration and service oriented application development.

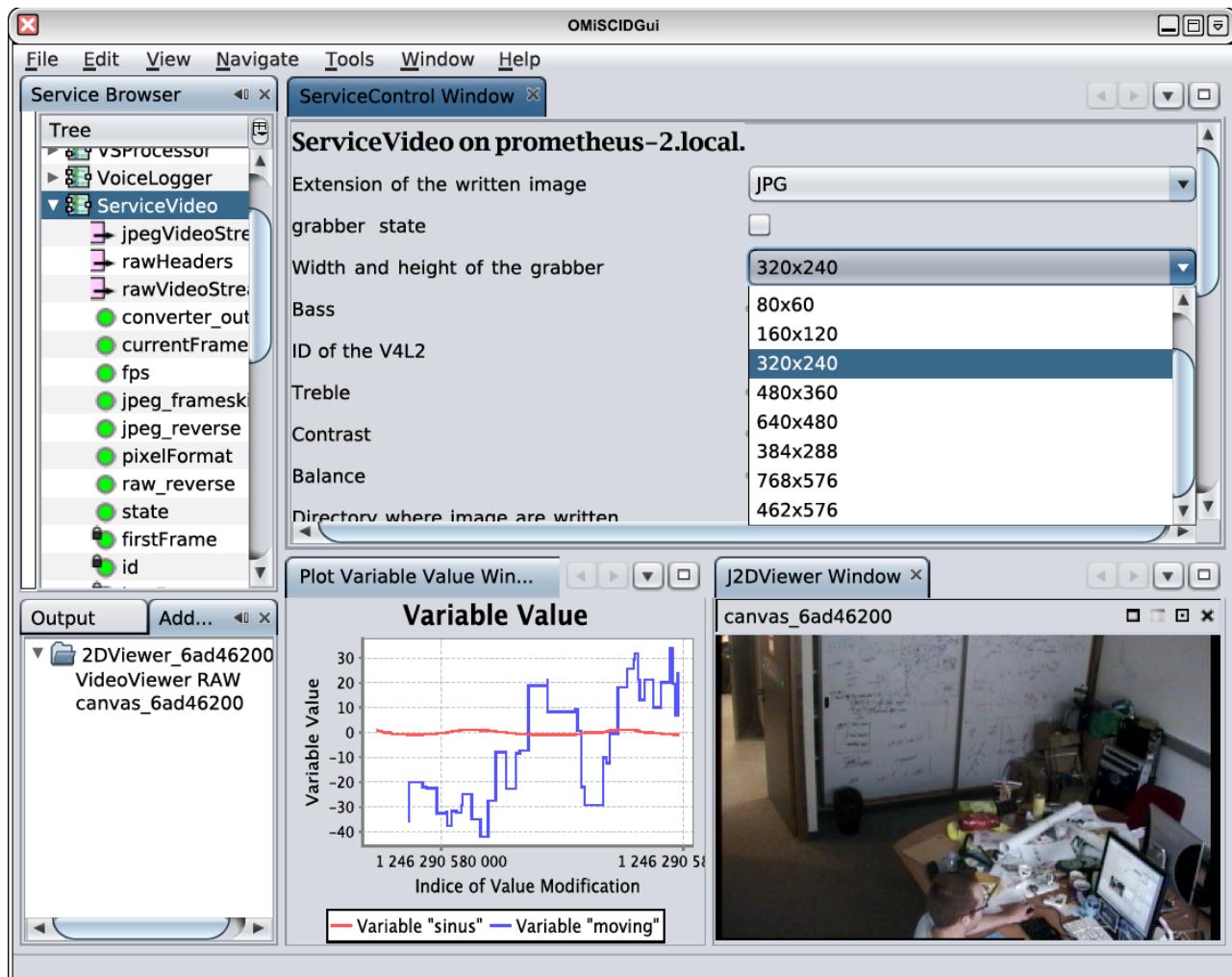


Figure 2: OMiSCID Gui. Public available plugins like running services properties (top left corner), variable value plotter (middle bottom), camera view (right bottom), service video controller (top right) are depicted.

IV. BRIEF TECHNICAL DESCRIPTION

One crucial requirement when designing a middleware for such heterogeneous research area is to make it usable by most of the people involved.

A. Multiplatform/Cross-Language

OMiSCID was designed with cross-platform/cross-language capabilities in mind. There are actually 3 supported implementation of OMiSCID: C++, Java and Python (PyMiSCID, note that the Python version used to be a simple wrapper around the C++ one). Moreover, the Java version can be used from Matlab and any other language running on the Java virtual machine (JavaFX script, scala, groovy, JavaScript, etc.) We also provide an OSGi abstraction layer that exposes OMiSCID with standard OSGi paradigm. OMiSCID was developed over a set of guidelines rather than over strict specifications. All the implementations are fully written in the target language, thus ensuring

speed, reliability, close integration with data structures and programming paradigm.

All versions are fully cross-platform and works on Linux, Windows and OS X both 32 bits and 64 bits. The C++ version uses an abstraction layer that provides common system objects like sockets, threads, mutexes, etc. The Java version has been successfully used on portable devices like a PDA and on the Android platform. All implementations can interoperate with each other on any supported platforms.

B. User Friendly API

In order to simplify interpersonal communications between OMiSCID users, we developed a common *User Friendly API*. It was defined to be easy to learn, easy to use and portable in several languages. Indeed, concepts, methods and parameters follow the same API in C++, Java and Python. However, each implementation takes advantage of the language specificities and design patterns. The API

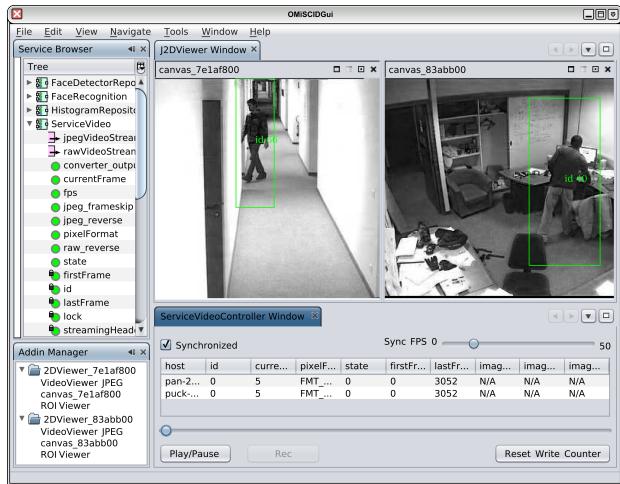


Figure 1: OMiSCID Gui session while monitoring and debugging multi-camera tracking system. Basic OMiSCID Gui functionalities are visible: monitoring of running services properties (top left corner), camera and tracking views (top right, available public plugins), video service controller in order to control 2 video services (right bottom) and the add-in manager to show active plugins (left bottom).

provides simple callback/listener mechanisms. Thus, one can be notified of many different events: a new connection from a service, a disconnection, a new message, a remote variable changed, etc.

The final *User Friendly API* is designed to be non-invasive. The OMiSCID code, inside a service, must be short, understandable and disjoined of the core source code. For instance, the following is a short example written in C++ that shows how to create a service with an output connector, how to register it over the network and finally how to send data to everyone connected to it:

```
#include <Omisid.h>
using namespace Omiscid;

// Create a service named camera
Service * pServ = ServiceFactory.Create( "Camera" );
// Add a output connector to it
pServ->AddConnector( "data flow",
                      "images stream", AnOutput);
// Register and start the service
pServ->Start();
// ...
// Send a video buffer to all client via
// the "video flow" connector
pServ->SendToAllClients( "video flow", Buffer );
```

C. New functionalities in OMiSCID 2.0

The current version of the OMiSCID middleware is 2.0. This version brings new requested functionalities: object serialization and remote procedure call.

Indeed, the philosophy of OMiSCID is to exchange information between services using either binary or textual

messages without any standard on the format of those messages. However it has been requested by developers to provide a simpler way to encode and decode textual messages. Thus, OMiSCID 2.0 provides a simple way to marshal and unmarshal any object to a JSON [17] representation. This allows easy communication between services without paying attention to the parsing and serialization of messages. The second improvement is the ability for a service to expose some of its functionalities by the means of remote callable methods. Such distant calls can be done either in a synchronous or in an asynchronous manner. Again these extensions are cross-platforms, cross-languages and benefit of specific improvement or features depending on the implementing language.

D. Performances

With the intention to provide our community with insight about OMiSCID performances, we conducted several experiments. Each one of these experiments were done on a cluster of computers called Grid5000 [18]. The choice of using a cluster is motivated by the fact that computers are on an isolated network, without any kind of perturbation (e.g., network maintenance, backup in progress, etc.). Moreover, it insures that each operating systems is newly installed, thus without side-effects of having old libraries or an unstable/tweaked system. The last criteria that motivate this choice is that experiments performed on Grid5000 are easily reproducible.

To evaluate the OMiSCID discovery process, we set up dedicated tests performing registering and searching tasks. Regarding the registering operation, at first sight, it is not a costless operation. Indeed we must generate and validate, for each newly created service, a unique service *id* (see Section III-A1) over the network. Our experiment shows that registering 100 services from 3 different computers takes less than 1 second. Regarding the searching task, also referred as lookup task, even if it is a linear process for OMiSCID (search time is linear in terms of number of running services to query), searching involves network communication and thus may take time. Our experiments reveal that finding a service among more than 400 others, distributed over 4 computers and using a simple variable value takes less than 20 ms long in average. Obviously, performing more specific searches, using for instance user-defined search filter (processing video stream to select a camera for instance) will eventually take more time.

Another performance measurement we performed was the latency introduced by OMiSCID message splitting mechanism (see Section III-B). Those tests were performed between two computers. We compared results using NTTCP—a Linux program that measures the transfer-rate between two computers, and using OMiSCID. Each test was run 1000 times using messages ranging from 512 bytes to 2 megabytes. The result shows that the latency introduced by

OMiSCID for message handling is around 4ms in average comparing to usual TCP/IP connections.

The reader might refer to [19] for other tests on performance and scalability.

V. CASES STUDY

For the past few years, we have used OMiSCID middleware in different research projects [20], [21], [19], [22]. In [19], OMiSCID is used to redesign a complete 3D Tracking system as well as an automatic cameraman. The redesign reduces the number of software components and has the advantage to provide shared and reusable services. For instance, both architectures use the same video grabbing services. This service provides real time streaming of camera images. This stream is simultaneously accessible by multiple services such as visualization services and image processing ones: movement detector, person detector, posture estimator.

OMiSCID middleware allows robustness for service discovery, reliable communication, connection and disconnection but also ease the federation of services. In [22], [21], OMiSCID is used for the implementation of a smart agent. The perception of the agent is provided by services dynamically discovered in the environment allowing the agent to construct a situation model [23] of the current situation. The agent is yet another service and, according to its perception, it is able to perform actions in the environment by sending orders to actuator services. In [22] the knowledge of the agent is distributed and can be stored on remote database using a combination of OSGi and OMiSCID. Each service developed is a reusable piece of software, which, by extension, ensures a decrease of development time along the years.

In the following sections, we will introduce a 3D video tracker—an OMiSCID redesigned example, some examples illustrating reusability (e.g., a network of Multi-modal Towers and Human Simon Game). The last illustration, present a Wizard of Oz experiment conducted in our lab which show various usage and benefits of using OMiSCID.

VI. SHOWCASE: REFACTORING, REUSABILITY AND MONITORING

A. Refactoring, the 3D Video Tracker case

To interact with people in smart environment, it is important to be able to detect people as well as to maintain an estimation of their current state e.g., position, speed, posture, etc. We have developed a video tracking system adapted to dynamic and complex environment. The 3D tracking was initially an improvement over an existing 2D tracker. It was running multiple 2D trackers and, by merging the different outputs, was able to compute a pseudo 3D estimation. This tracker had initially a monolithic architecture for performance reason: image acquisition and tracking process were done within a single process.

A full 3D tracker must estimate directly the targets in 3D using information from several cameras. Evolution from 2D to 3D tracking required a change in the architecture as the complexity became too high to run on a single processor. For instance, the number of camera increasing from one to four and sometimes many more cameras. For robustness, reusability and maintainability reasons it is mandatory to split image acquisition and processing software part.

The chosen architecture is shown in Figure 3. This architecture introduces a new concept: service factory. Services factories are services (following the factory method pattern) that are designed to start but also instantiate, on demand, a (parametric) service. Factories can be seen as daemon services, ready to launch other service(s) as to fulfil applications needs. Such scheme eases the deployment of distributed applications. In this example, a Video Service Processing Factory can create a Video Service Processing (VSP) with special pipeline treatments over images from a camera.

The architecture is thus modular and distributed in order to reduce computation time and to avoid costly images transfer over the network:

- A video service is in charge of grabbing images from each camera;
- A video service processor (VSP) factory is attached to each video service. Its role is to initialize one or more VSP in charge of image processing tasks;
- The main tracker program is in charge of targets tracking; it automatically detects all VSP Factories, requests for VSP creation and connects to them.

This refactoring, performed for the 3D tracker, allows us to create lots of software components that are reused later in many other perceptive applications.

Demonstrations of the 3D tracker system are available on the PRIMA channel (see the Tracker, Human Simon Game and more videos on the PRIMA channel [24]).

B. Reusability, the Multimodal-Tower Network case

In the context of the CASPER project (Communication, Activity Analysis and Ambient Assistance for Senior PERsons), a project for maintaining elderly people at home, we developed a multi-modal localization system. This system (see Figure 4) associates on each tower an omnidirectional camera with an array of microphones. Tracking is done combining visual tracker of bodies and acoustic tracker of people when they speak.

Building such application is facilitated by the reusability of already available services: microphone service for audio recording, speech activity detection service, video service for images acquisition, VSP factory (see previous section) for video processing. Only one service was built in this case, the localization service. This service can dynamically register, using filters (see Section III-C), to all data coming from all

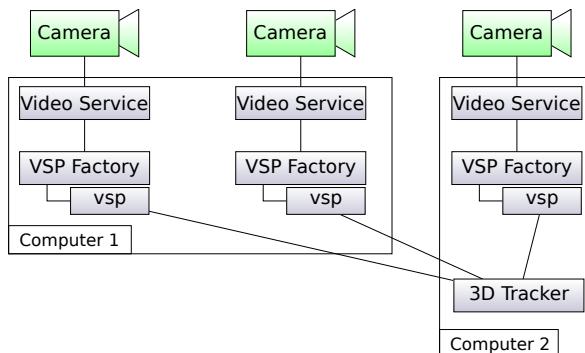


Figure 3: Video tracker architecture. For each camera, a Video Service is in charge of grabbing images and sends them to a Video Service Processor (VSP). This Video Service Processor is instantiated on demand by a VSP Factory. The 3D tracker service automatically finds all VSP Factories, asks them to create VSP and connects to them as to handle targets management.

towers in the room. Then, after an auto-calibration phase, it can track speaking targets in the room.

As it is usually the case, reusability increases maintainability. Using services in many different contexts is a good way to deeply test them in adverse conditions, with new client services. For instance, in the realisation of this system, we highlighted different issues in certain existing services that were not evident to spot before their integration in this particular setting. It is not worth to say that, correcting problems in a service is a benefit for all applications using it.

C. Reusability, the Human Simon Game case

Validating performances of a 3D video tracking system is usually done using well known databases containing annotated recordings providing ground truth labels. The results of the evaluated system are then compared to the ground truth using different kinds of metrics and thresholds in order to provide a score. Such approach are very convenient to validate a tracking system regarding the state-of-the-art (for publication purpose for example). However, generally, the tests are performed off-line and the threshoulds often lead to imprecise measurements. Additionally, test-databases are usually designed to evaluate very specific characteristics and are conditioned as well by very specific environmental conditions. As a result, the outcomes obtained by testing your system over such databases are not necessary representative of the real performance of your system. They might not highlight the particular benefits your system is bringing compared to other existing solution. Robustness to environmental artefacts, such as change in light exposure or random ambient noise for instance, is often omitted. In an attempt to provide an online alternative for the validation of our 3D video tracking



Figure 4: Multimodal tower equipped with an omnidirectional camera and a set of microphones. Multiple towers may be disseminated in the environment and constitute a sensory-network used to follow and infer activity of elderly people at home. One can see the tower itself, a view from one panoramic camera (top right) and the auto-calibration algorithm configuring the relative position of 3 towers.

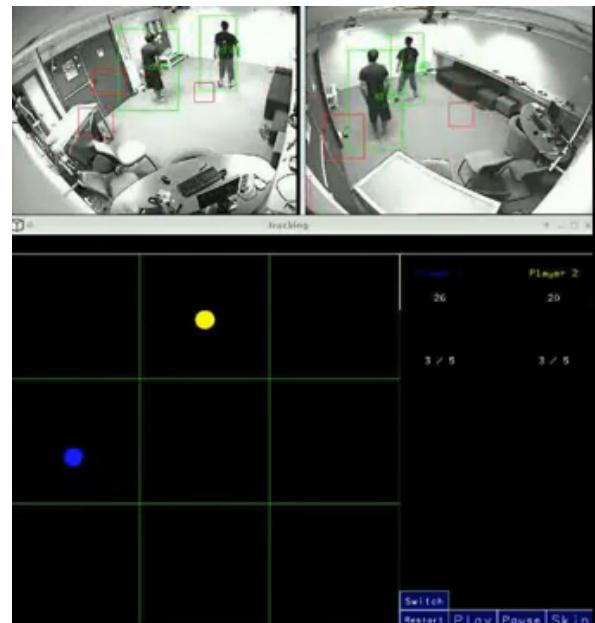


Figure 5: Human Simon Game. Players follow projected instructions on the wall. They play against each other. They must run and squat in cells to activate them. As in usual simon game, the activation sequence is longer each turn¹⁰.

system, we designed an experimental-game showing, in real time, the performance (speed performance, number of false detections, resistance to occlusions, etc.) of our solution in adverse conditions (rapid moves, occlusions, bad lightning condition, etc.).

In order to achieve this goal, we designed a Human Simon Game (see video on the PRIMA channel [24]) based on the famous game for child: one need to remember an audio sequence associated to blinking color buttons. In our case, as seen on Figure 5, we decided to virtually cut the room space in 9 cells, each representing a "blinking button" of the Simon game. Cells can be activated by one or two players squatting and standing back up in it. The game is similar to the usual Simon Game but is adapted for two players: each player has a color and must validate an always increasing sequence of cells by moving and squatting over the virtual 3x3 grid.

In our case, setting up this validation system was an easy task. The 3D video tracker, the posture estimator (for squats), the steerable projector (projecting the game interface on a wall in front of the players) were all existing services that we reused. The only software component developed was the core algorithm of the actual game.

VII. FULL CASE STUDY: WIZARD OF OZ EXPERIMENT

In the field of human-computer interaction, a Wizard of Oz (WOz) experiment is a research experiment, in which subjects are confronted with a computer system that subjects believe to be autonomous, but that is actually being operated or partially operated by some hidden experimenter(s): the wizard(s). The goal of such experiments is to study the usability, acceptability and efficiency of a proposed system, functionality or interface—often hypothetic or unfinished—by evaluating the interaction of the subjects with it rather than focusing on the quality of the proposed solution. The advantages of performing a WOz experiment versus actually evaluating the real system or interface are that it saves time and money. Indeed the WOz experiment is a quick and cheap way of (in)validating a set of proposed functionalities without investing resources in a system that might be, first, reconsidered regarding the feedbacks collected from the user experience, and second, just impossible to design due to the absence of required technologies to build it, or the lack of budget and/or time. In the WOz, certain functionalities can be implemented while reconsidering the other. For instance, if we consider the case of smart environments—environment equipped with sensors and actuators designed to provide assistance to user in daily tasks and activities, it can be very annoying and sometimes not an option, to dispose of a perfectly working environment if the only functionality wanted to be evaluated is, for instance, how users manage to undo an action performed in the background by this environment—switching the TV off, turning the music back on, opening the shutter or raising up/down

the volume of some other devices. A more time-efficient and money-efficient option would be to focus the resources on the functionalities of interest and manage to fake the other functionality by, for instance, remote controlling the environment by one or more experimenter. In a WOz experiment, the missing functionalities are emulated by an experimenter hidden from the subjects performing the evaluation. In most settings, subjects are located in a room along with the system to evaluate while the wizard operates in another room. Both rooms can be separated by a beam splitter allowing the wizard to observe and react accordingly to the subject(s) actions.

A. Requirements

Even if performing a Wizard of Oz (WOz) remains in many ways more advantageous, depending the context of the experiment, setting up such experiment requires an important preparation. Importantly, additional constraints appear if the settings have to be mobile, i.e., to be carried in different places. The wizard must have access to a multitude of information in order to control the system as well as possible. Without the presence of a beam splitter, the environment must be equipped with cameras, microphones and speakers to record and stream the scene in real time. Among those devices, the wizards (there might be more than one), also need the proper controllers to remotely manipulate the system. Such a setting requires an extensive use of wireless or wired communication between software components: controllers and controles. In addition, the coupling between software components has to be able to change and to be easy to achieve. Allowing for instance to deploy debuggers, loggers or visualization tools at runtime. The more reusable the perceptual/actuator software components are, the cheapest and fastest the experiments will be.

B. Experimental Settings

On an ongoing research project [25], we sought to evaluate the behavior of subjects immersed in a ubiquitous environment while asked to teach a smart agent how to control the space. Among few, the objectives were to validate hypothesis about human-machine interaction as well as to collect constructive outcomes that will help future design of ambient systems.

Four kinds of actor are to be considered in this experiment: the subjects, the smart agent, the environment and the wizards. The *subjects* by group of 2 or 3, are asked to teach the agent to control the environment in order to organize a small meeting. A classic example would be for the subjects to teach the agent to switch on a light when people are entering the room, and, to switch it back off when everybody is leaving.

The *agent* is embodied by a personal mobile phone with wireless capabilities, on which we deployed a learning

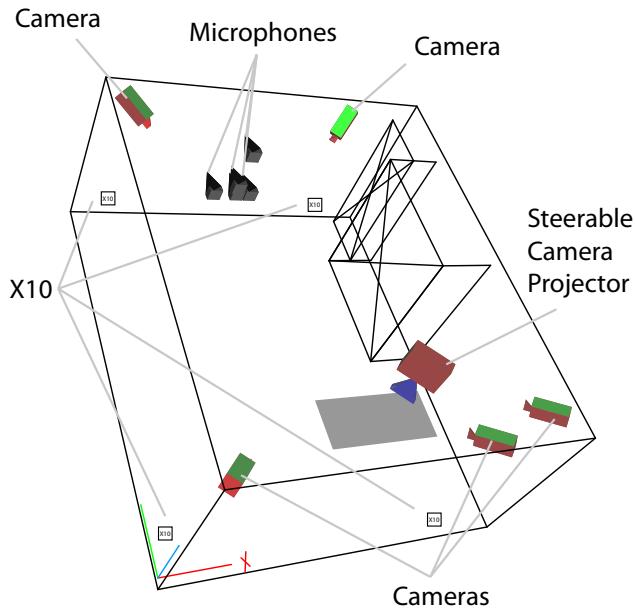


Figure 6: PRIMA's Smartroom. The smartroom is equipped with perceptive equipments (cameras and microphones), and with actuators (mobile steerable camera/projector pair, X10 power controllers). It has been designed to permit immersive user studies with activity recognition (perception and situation modelling) and system feedbacks.

software and a simple user interface. This interface allows collecting real-time feedbacks from the subjects (good, bad) during the session. Alternatively the agent can be embodied by other devices such as, for instance, a robot. In this case, more perceptual components are available for the experimenters and the interaction between the subjects and the agent is different (e.g., the rewards are more natural to provide). The agent, as any other service, connects through the wireless network to a situation modeller service that provides a situation model [21] of the current situation. When requested by the subjects, the agent performs an action in the environment by sending orders to actuator services present in the environment. Subjects can, whenever they agree or disagree, give a positive or negative reward to the agent. With such a settings, the agent learns to control the environment, senses and acts using dynamically discovered services, and finally, learns from the feedback provided by the subjects the correct association between situations and actions.

The *environment* is an office (Figure 6) spread with many actuators and sensors. OMiSCID allows each of them to be accessible and controllable by services all-over the network (Figure 7). Among the sensors we find: cameras, microphones, a thermometer and a weather station. All the actuators are controllable by OMiSCID connectors and their

states can be queried by those connectors or are exposed through variables. Among the actuators, we list: a steerable video projector, some x10 controllers, loud speakers and even windows shutter.

For this experiment, we needed two wizards. The *first wizard* was in charge of simulating certain actuators in the environment such as pressure detectors under the chairs and sofas. Indeed, it was told to the subjects that each chair was mounted with pressure detector to detect when someone sits. Because we had no such device installed in our environmental facility, we emulated those actuators using a user interface plugged into OMiSCID Gui. The simulating interface was seen as yet another service that can be used by the situation modeler to build up a better situation model. The *second wizard* was controlling the overall experiment using a master interface. This interface allowed writing real-time observations through an annotator service, as well as taking control over all the services in the environment. Such a master control for instance let the wizard speed up the experiment by helping the agent to guess better actions (when subjects got exhausted).

C. OMiSCID At Glance

For this experiment we deployed more than 20 services spread on 5 computers running different operating systems (Linux, Windows and MacOSX). Figure 7 presents some of the devices present in the environment as well as the interconnection of services. Due to the complexity of the schema some services have been removed. We next review some of the advantages of using OMiSCID in this experiment:

1) *Multi-platform*: 5 computers have been used during the experiments, two of them by the wizards. One of the wizards was using MacOS, on which we deployed the master control. Due to driver issue the sound recording system was using a Microsoft powered computer. The video streaming as well as all the other services (archiver, x10, etc) were running on Linux hosts.

2) *Multi-language*: To design the services, we have used different languages. C++ was used for performance reasons such as for the video and sound processing/capture services. Python is a really powerful language for the rapid prototyping of application. We used Python to quickly develop the x10 or the PanTilt controllers. Java has been used to develop some of the OMiSCID Gui module but also to access the different online web services exposed in the environment such as the weather service. JavaFX was used to develop the wizard control's interface. Its script language makes it easy to use for inexpensive user interface design.

3) *Service Discovery*: The simple but powerful service discovery system provided by OMiSCID has been used to dynamically connect services together. The best examples are the *situation modeler* and the *archiver*. Using a service repository, they were able to filter services that were present

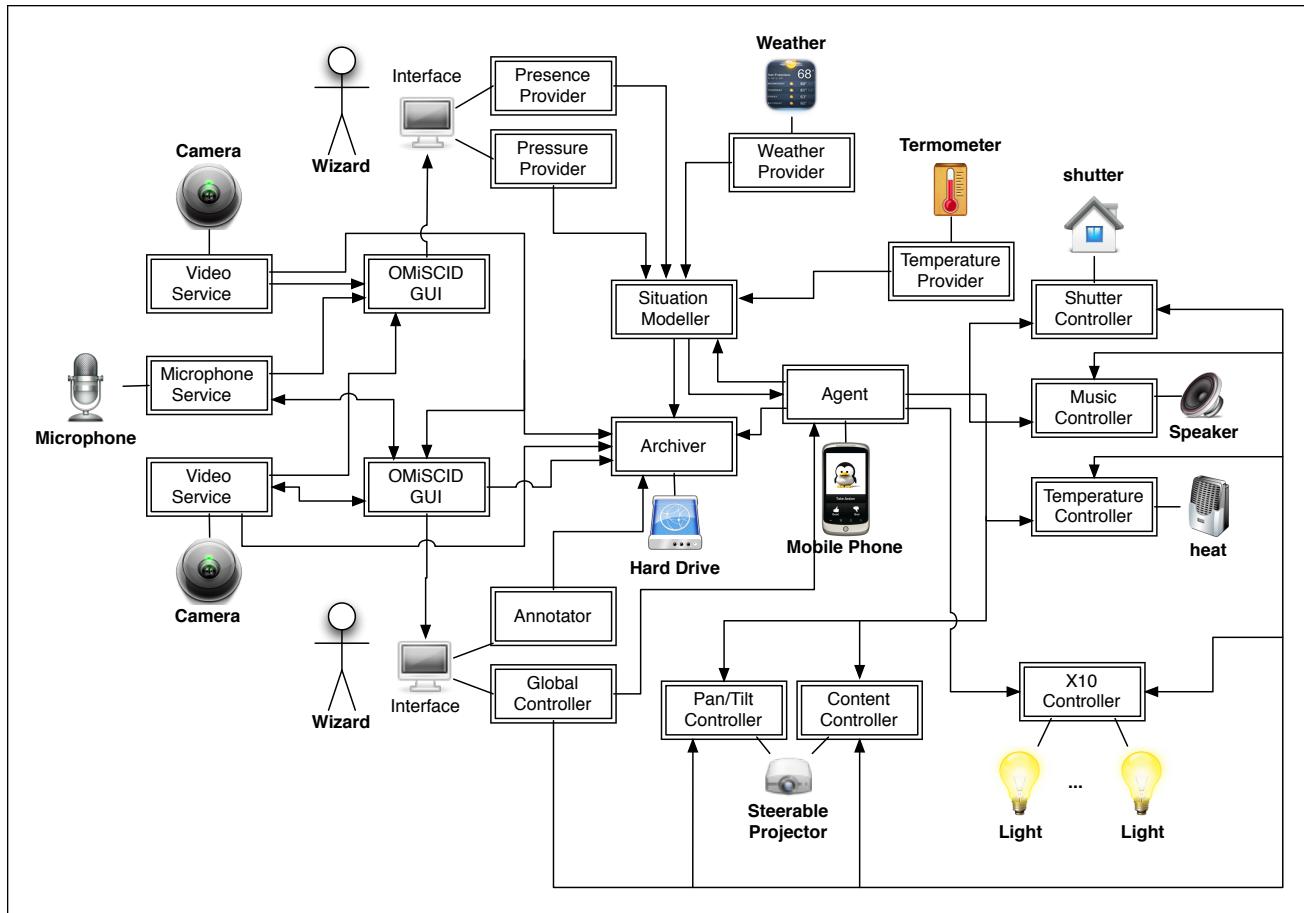


Figure 7: Graph of services for our Wizard of Oz experiment. Services used were mostly created for other experimentations (video services, microphone service, music controller, etc.) and were directly reused and interconnected with specific orchestrating services for this experiment (annotator, global controller).

in the environment in order to connect to them. For instance the *situation modeler* was looking for all services having connector or variable exposing state information. Using that state information, it was able to provide a situation model on an output connector. The *archiver* was responsible to backup any information transmitted between services on a hard drive. The archiver was continuously looking for all services having output connector. Thus it was easy for instance to deploy or shutdown services on the fly during the experiment.

4) Communication: Communication between services was achieved using different format. For video and sound services, data were raw binary information tagged with time stamps. Web services such as the weather provider were communicating information using XML on their connector. The PanTilt controller exposed its commands by the means of remote callable methods, and presented its internal state using readable variable.

5) OMiSCID Gui: OMiSCID Gui was used by the wizard for different purpose. Firstly, the streamed sound and video

were played by the embedded player. Indeed, we have developed OMiSCID Gui modules to play video and listen to audio stream in real-time. Those modules have been used to get a feedback of what was happening into the experimental facility disposed into another building. Secondly, OMiSCID Gui was used to control the archiver and other services.

6) Reusability: Each of the service used in this experiment is a reusable piece of software that can be carried and deployed easily. For a wizard of Oz experiment, only the hardware and the equipments (cameras, microphones) have to be transported and reinstalled. Everything else is deployable instantly and can adapt to the configuration: number of computers, operating systems, number and nature of devices, etc.

VIII. CONCLUSION

OMiSCID is an efficient and lightweight solution for the rapid prototyping of Service Oriented Architectures and applications in the context of ubiquitous computing and ambient intelligence. The solution provides a user-friendly

API to declare, to describe, to discover and to interconnect services as well as to manage their communications.

To be attractive, OMiSCID offers to researchers several facilities for ubiquitous computing with its multi-platform, cross-language capabilities and interoperability. The underlying concepts as well as the API are user friendly and directed toward usability, extensibility, reusability and maintainability. In contrast to existing solutions, the API is non-invasive which keeps the developed solution portable to other paradigm if mandatory. OMiSCID can also be mixed or extend other middleware solutions as we did with the OSGi platform. Using OMiSCID for service discovery, one can built an ubiquitous application interconnecting dozens of services. All networking aspects are handle by the middleware as this solution does not rely on the number of computers, operating systems or network configuration. One can concentrate about core services that will orchestrate the full application. As transfer-rate performances are not altered by OMiSCID usage, it is possible to transfer data from a simple integer to huge video streams. Target applications are thus not limited by OMiSCID.

Along with this middleware, OMiSCID Gui provides developers with an extensible, portable and modular platform that ease development and debugging, and improve maintainability of OMiSCID demonstrations and applications.

OMiSCID has successfully been used in several academic research projects and more recently in a wizard of Oz experiment. Such an experiment requires an important amount of resources and preparations, particularly when realized in smart environments. We have presented how OMiSCID and OMiSCID Gui can greatly reduce development time, maximizing reusability of existing software, and eases redeployment.

IX. ACKNOWLEDGEMENT

For their past work on OMiSCID and/or experimentations depicted in this article, the authors would like to thank (in anti-chronological order) Wafa Benkaouar, Matthieu Langet, Jean-Pascal Mercier, Julien Letessier and Sébastien Pesnel.

REFERENCES

- [1] R. Barraquand, D. Vaufreydaz, R. Emonet, and J. Mercier, "UBICOMM 2010 paper Case Study of the OMiSCID Middleware: Wizard of Oz Experiment in Smart Environments," in *The Fourth International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies*, Florence, Italy, 2010.
- [2] R. Emonet, D. Vaufreydaz, P. Reignier, and J. Letessier, "O3misd: an object oriented opensource middleware for service connection, introspection and discovery," in *1st IEEE International Workshop on Services Integration in Pervasive Environments*, Lyon (France), jun 2006.
- [3] "OSGi Alliance," accessed 17-January-2012. [Online]. Available: <http://www.osgi.org/>
- [4] C. Escoffier, R. S. Hall, and P. Lalande, "ipojo: an extensible service-oriented component framework," *Services Computing, IEEE International Conference on*, vol. 0, pp. 474–481, 2007.
- [5] D. Wang, L. Huang, J. Wu, and X. Xu, "Dynamic software upgrading for distributed system based on r-osgi," in *CSSE '08: Proceedings of the 2008 International Conference on Computer Science and Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 227–231.
- [6] "Referenced specifications UPnP forum," accessed 17-January-2012. [Online]. Available: <http://upnp.org/sdcps-and-certification/standards/referenced-specifications/>
- [7] "OW2 chameleon," accessed 17-January-2012. [Online]. Available: <http://wiki.chameleon.ow2.org/xwiki/bin/view/Main/Rose>
- [8] C. Escoffier, J. Bardin, J. Bourcier, and P. Lalande, "Developing User-Centric Applications with H-Omega," in *Mobile Wireless Middleware, Operating Systems, and Applications - Workshops*. Springer Berlin Heidelberg, April 2009, pp. 118–123.
- [9] M. Papazoglou, *Web Services: Principles and Technology*. Prentice Hall, September 2007.
- [10] R. Khalaf, N. Mukhi, and S. Weerawarana, "Service-oriented composition in bpel4ws." in *WWW (Alternate Paper Tracks)*, 2003.
- [11] D. Martin, M. Paolucci, S. McIlraith, M. Burstein, D. Mcdermott, D. McGuinness, B. Parsia, T. Payne, M. Sabou, M. Solanki, N. Srinivasan, and K. Sycara, "Bringing semantics to web services: The owl-s approach," in *SWSWPC 2004*, ser. LNCS, J. Cardoso and A. Sheth, Eds., vol. 3387. Springer, 2004, pp. 26–42.
- [12] A. Fillinger, L. Diduch, I. Hamchi, M. Hoarau, S. Degre, and V. Stanford, "The nist data flow system ii: A standardized interface for distributed multimedia applications," in *World of Wireless, Mobile and Multimedia Networks, 2008. WoWMoM 2008. 2008 International Symposium on a*, 23-26 2008, pp. 1 –3.
- [13] "OMiSCID Forge," accessed 17-January-2012. [Online]. Available: <http://omiscid.gforge.inria.fr/>
- [14] "The official bluetooth SIG member website | specification: Adopted documents," accessed 17-January-2012. [Online]. Available: <https://www.bluetooth.org/Technical/Specifications/adopted.htm>
- [15] D. Svensson Fors, B. Magnusson, S. Gesteg\aaard Robertz, G. Hedin, and E. Nilsson-Nyman, "Ad-hoc composition of pervasive services in the PalCom architecture," in *Proceedings of the 2009 international conference on Pervasive services*, ser. ICPS '09. London, United Kingdom: ACM, 2009, p. 83–92, ACM ID: 1568213.
- [16] "YAML on Wikipedia," accessed 17-January-2012. [Online]. Available: <http://en.wikipedia.org/wiki/YAML>
- [17] "JSON Website," accessed 17-January-2012. [Online]. Available: <http://www.json.org/>

- [18] F. Cappello, F. Desprez, M. Dayde, E. Jeannot, Y. Jégou, S. Lanteri, N. Melab, R. Namyst, P. Primet, O. Richard, E. Caron, J. Leduc, and G. Mornet, "Grid'5000: a large scale, reconfigurable, controlable and monitorable Grid platform," in *6th IEEE/ACM International Workshop on Grid Computing - GRID 2005*, Seattle, USA, 11 2005.
- [19] R. Emonet, "Semantic description of services and service factories for ambient intelligence," Ph.D. dissertation, Grenoble INP, sep 2009.
- [20] J. L. Crowley, D. Hall, and R. Emonet, "Autonomic computer vision systems," in *Advanced Concepts for Intelligent Vision Systems, ICIVS 2007*, J. Blanc-Talon, Ed. IEEE, Eurasip, Aug 2007.
- [21] R. Barraquand and J. L. Crowley, "Learning polite behavior with situation models," in *HRI '08: Proceedings of the 3rd ACM/IEEE international conference on Human robot interaction*. New York, NY, USA: ACM, 2008, pp. 209–216.
- [22] S. Zaidenberg, P. Reignier, and J. L. Crowley, "An architecture for ubiquitous applications," *Ubiquitous Computing and Communication Journal (UBiCC)*, vol. 4, no. 2, jan 2009.
- [23] J. L. Crowley, P. Reignier, and R. Barraquand, "Situation models: A tool for observing and understanding activity," in *Workshop People Detection and Tracking, held in IEEE International Conference on Robotics and Automation*, Kobe, Japan, 2009.
- [24] "PRIMA Channel on Youtube," accessed 17-January-2012. [Online]. Available: <http://www.youtube.com/user/PrimaChannel>
- [25] R. Barraquand, P. Reignier, and N. Mandran, "The Sorceress of Oz," in *Workshop for Pervasive Intelligibility, part of the Pervasice Conference*, San Francisco, USA, 2011.

On a New Method for Derivative Free Optimization

Lennart Frimannslund

*Department of Informatics
University of Bergen
Bergen, Norway*

Email: lennart@ii.uib.no

Trond Steihaug

*Department of Informatics
University of Bergen
Bergen, Norway*

Email: trond.steihaug@ii.uib.no

Abstract—A new derivative-free optimization method for unconstrained optimization of partially separable functions is presented. Using average curvature information computed from sampled function values the method generates an average Hessian-like matrix and uses its eigenvectors as new search directions. Numerical experiments demonstrate that this new derivative free optimization method has the very desirable property of avoiding saddle points. This is illustrated on two test functions and compared to other well known derivative free methods. Further, we compare the efficiency of the new method with two classical derivative methods using a class of testproblems.

Keywords-Generating Set Search, Derivative-Free Optimization, Saddle points, Sparsity.

I. INTRODUCTION

Continuous optimization is an important area of study, with applications in statistical parameter estimation, economics, medicine, industry — simply put, anywhere a mathematical model can be used to represent some real-world process or system which is to be optimized. Mathematically, we can express such a problem as

$$\min_{x \in D \subseteq \mathbb{R}^n} f(x), \quad (1)$$

where f is the objective function, based on the model which is defined on the domain D . These models can range from simple analytic expressions to complex simulations. Well known optimization methods such as Newton's method use derivatives to iteratively find a solution. These derivatives must somehow be provided, either through explicit formulas/computer code, or, for instance, automatic differentiation.

Suppose, however, that the objective function is produced by some sort of non-differentiable simulation, or that it involves expressions which can only be computed numerically, such as the solution to differential equations, integrals, and so on. In this case derivatives might not exist, or they may be unavailable if the numerically computed function is subject to some kind of adaptive discretization and truncation and therefore is non-differentiable, unlike the underlying mathematical function. In these cases derivative-based methods are not directly applicable, which leads to the need of methods that do not explicitly require derivatives.

For an introduction to derivative free methods the reader is referred to [3].

Generating set search (GSS) methods are a subclass of derivative-free methods for unconstrained optimization. These methods can be extended to handle constraints, but we will focus on the unconstrained case where the domain D in the problem (1) is equal to \mathbb{R}^n . A comprehensive introduction to these methods can be found in [14]. In their most basic form these methods only use function values and do not collect any information such as average slope or average curvature information. Computing this information, however, can significantly speed up convergence, and this is done in the methods presented in [4], [6].

In addition, information about the structure of the function known a priori can also be useful. Suppose that the objective function f can be written as a sum of element functions,

$$f = \sum_{i=1}^m f_i,$$

where each element function has the property that it is unaffected when we move along one or more of the coordinate directions. For example, we might have

$$f(x_1, x_2, x_3) = f_1(x_1, x_2) + f_2(x_2, x_3). \quad (2)$$

Then, the function is said to be partially separable [10] and we say that f_i has a large null space. If f is partially separable and twice continuously differentiable, then its Hessian matrix,

$$\nabla^2 f(x) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \vdots & & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix},$$

will be sparse. For the function (2) the Hessian element $\frac{\partial^2 f}{\partial x_1 \partial x_3}$ will be zero. If the function (2) is not twice continuously differentiable, then the matrix of the corresponding finite differences, that is, the matrix with

$$\begin{aligned} & \left[f(x_1 + h, x_2, x_3 + k) - f(x_1 + h, x_2, x_3) \right. \\ & \left. - f(x_1, x_2, x_3 + k) + f(x_1, x_2, x_3) \right] / (hk) = 0, \quad (3) \end{aligned}$$

in position $(i, j) = (1, 3)$ (and with similar expressions for all other (i, j) -pairs) will be sparse for any x , and any nonzero h and k , none of which have to be the same for each (i, j) -pair. The sparsity structure is the same as for the differentiable case, so that the expression (3) is identically zero. This result can be extended to any partially separable function, as proved in [7].

In [23], a GSS method which exploits such structure is presented, which is applicable to the case where these element functions are individually available.

In this paper, we present a GSS method which takes advantage of the partially separable functions, without requiring the element functions (which may or may not be differentiable) to be available. It is an extension of the paper [6]. We use the concept of average curvature introduced in [6].

This paper is organized as follows. In section II, we outline a basic framework for GSS, as well as the previous work of the authors on which the present paper is based. In Sections III and IV, we present the handling of partially separable functions and the convergence to second order stationary points. Section V contains a discussion of the methods used in the comparison and Section VI specifies the test functions. The main part of this paper is the testing presented in Section VII. Here, we define region of convergence and in the two sections VII and VIII we present the numerical properties of the methods on the two test functions. In Section IX, we show the efficiency of method derived in this paper compared to two classical methods for derivative free optimization. Concluding remarks are given in Section X.

II. GENERATING SET SEARCH USING CURVATURE INFORMATION

We restrict ourselves to a subset of GSS methods, namely sufficient decrease methods with $2n$ search directions, the positive and negative of n mutually orthogonal directions, of unit length. These directions will in general *not* be the coordinate directions. A simplified framework for the methods we consider is given in Figure 1. The univariate function ρ must be nondecreasing and satisfy $\lim_{x \downarrow 0} \frac{\rho(x)}{x} = 0$. For simplicity, increasing the step length can be thought of as multiplying it by 2, and decreasing it as dividing by 2, although these rules may be more advanced. For the formal requirements on these rules, see [14]. Given mild requirements on the function f the step length δ will ultimately go to zero, and the common convergence criterion for all GSS methods is that δ is smaller than some tolerance.

Given set of search directions \mathcal{Q} , step length δ and an initial guess $x \leftarrow x_0$.

While δ is larger than some tolerance

Repeat until x has been updated or all $q \in \mathcal{Q}$ have been used:

Get next search direction $q \in \mathcal{Q}$.

If $f(x + \delta q) < f(x) - \rho(\delta)$

Update x : $x \leftarrow x + \delta q$.

Optionally increase δ .

End if

End repeat

If no search direction provided a better function value, decrease δ .

Optionally update \mathcal{Q} .

End while

Figure 1. Simplified framework for a sufficient decrease GSS method.

As can be seen from the pseudo code in Figure 1, the set of search directions can be periodically updated. In [6], the authors present a method that computes average curvature information from previously sampled points, assembles this information in a Hessian-like matrix and uses the eigenvectors of this matrix as the search directions, which amounts to a rotation of the old search directions. Once this rotation has been performed, the process restarts, and new curvature information is computed, periodically resulting in new search directions. It is shown that the efficiency of the method can be greatly improved compared to just using the coordinate directions as the search directions throughout.

A similar scheme, which aligns the basis to the average direction the search progresses, appeared as early as 1960 in [24] and implemented in 1973 [16]. To illustrate the idea of curvature information we use a quadratic model function by assuming we are minimizing, say,

$$g(y) = \phi + b^T(y - x) + \frac{1}{2}(y - x)^T C(y - x),$$

where C is a symmetric matrix. The search directions are positive and negative of the column vectors of the orthogonal matrix Q , that is,

$$Q = \begin{bmatrix} q_1 & q_2 & \cdots & q_n \end{bmatrix}.$$

Since g is a quadratic function, we have

$$q_i^T C q_j = \frac{g(x + \delta_i q_i + \delta_j q_j) - g(x + \delta_i q_i) - g(x + \delta_j q_j) + g(x)}{\delta_i \delta_j}.$$

For a general function f the computation of curvature information can be done in the following way, which is a slight modification of the methodology presented in [6]. Consider Figure 2, and assume that the current point is the

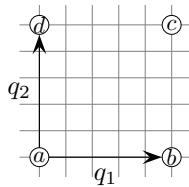


Figure 2. Location of sampled points used for curvature computation.

Outcome	Notes
SS	The search along q_1 moves the current best point to b , and the search along q_2 moves the current best point to c . The function value at d must be computed separately.
SF	The search along q_1 moves the current best point to b , and the search along q_2 computes the function value at c , but does not move the current best point. The function value at d must be computed separately.
FS	The search along q_1 computes the function value at point b , but does not move the current best point. The search along q_2 computes the function value at point d . The function value at point c must be computed separately.
FF	Neither the search along q_1 nor q_2 update the current best point, but the function values at points b and d are obtained. The function value at point c must be computed separately.

Table I

THE FOUR POSSIBLE OUTCOMES WHEN SEARCHING ALONG TWO CONSECUTIVE DIRECTIONS. S MEANS SUCCESS, F MEANS FAILURE.

point marked a , and that the next two search directions in the repeat-loop in the pseudo code are the directions shown, q_1 and q_2 . When searching along two directions in a row, there are four possible outcomes. Success-success (both the search along q_1 and q_2 produce function values which satisfy the sufficient decrease condition), success-failure (the search along q_1 produces a sufficiently lower function value, but the search along q_2 does not), failure-success, and finally failure-failure. In all of these four cases, by computing the function value at a fourth point, the function values at four points in a rectangle can be obtained. The details are given in Table I. The function values at four such points a , b , c and d can be inserted into the formula

$$\frac{f(c) - f(b) - f(d) + f(a)}{\|b - a\| \|d - a\|}. \quad (4)$$

If the objective function is twice continuously differentiable, then the next lemma will show that (4) is equal to $q_1^T \nabla^2 f(\hat{x}) q_2$, where \hat{x} is some point within the rectangle $abcd$. If the function is not twice continuously differentiable, (4) captures the average curvature in the rectangle.

The rectangle lies in the plane spanned by the search directions q_1 and q_2 since these were used consecutively. By successively reordering how the “get next search direction”

statement considers the directions in \mathcal{Q} , one can obtain curvature information with respect to all the $n(n-1)/2$ possible different combinations of search directions, in a finite and uniformly bounded number of steps, which depends on n since there are $O(n^2)$ elements of curvature information which must be assembled. (For this reason, the method is not suitable for n larger than about 30, but exploiting structure can allow for much larger n , as will be explained in Section III.)

The following lemma is a slightly modified version of [5, Lemma 3.5] and can be found in calculus textbooks usually as a part of showing that the Hessian matrix is symmetric if the function is sufficiently smooth.

Lemma 1: Suppose the objective function $f : \mathbb{R}^n \mapsto \mathbb{R}$ is twice continuously differentiable, assume we have given two orthogonal search directions q_i and q_j , and have computed

$$f(x), f(x + hq_i), f(x + kq_j), \text{ and } f(x + hq_i + kq_j)$$

for some x and some scalars h and k . Let element ij , $i > j$ of the symmetric matrix C_Q be

$$(C_Q)_{ij} = \frac{f(x + hq_i + kq_j) - f(x + hq_i) - f(x + kq_j) + f(x)}{hk}.$$

Then,

$$(C_Q)_{ij} = q_i^T \nabla^2 f(\hat{x}) q_j,$$

where $\hat{x} = x + \tau hq_i + \sigma kq_j$ for some $\tau, \sigma \in [0, 1]$.

The matrix C_Q contains $q_i^T \nabla^2 f(\hat{x}) q_j$ in positions (i, j) and (j, i) , which is curvature information with respect to the coordinate system defined by the n directions q_1, \dots, q_n in \mathcal{Q} . Note that the point \hat{x} is different for each (i, j) -pair. Also note that both $q_i \in \mathcal{Q}$ and $-q_i \in \mathcal{Q}$. The diagonal elements of C_Q must be computed separately, for instance when the step length is reduced, since the preceding repeat-loop, combined with the current f -value then gives the function values at three equally spaced points on a straight line for all n search directions.

Once the matrix C_Q is complete, it is subjected to the rotation

$$C = QC_Q Q^T, \quad (5)$$

where Q is the matrix with the n unique search directions as its columns, ordered so that they correspond to the ordering of the elements in C_Q . C now contains curvature information with respect to the standard coordinate system. The search directions in \mathcal{Q} are then replaced with the positive and negative of the eigenvectors of C .

To build up C_Q in a systematic fashion we need to specify one way to choose the order. For instance, for $n = 4$ and one wants to compute $(C_Q)_{21}$, $(C_Q)_{31}$, $(C_Q)_{24}$, and $(C_Q)_{34}$, then one can let the order of the directions be:

$$q_1, q_2, -q_1, q_3, -q_2, q_4, -q_3, -q_4.$$

Here, the search along q_1 and q_2 enables us to compute $(C_Q)_{21}$. The directions $-q_1$ and q_3 provide us with $(C_Q)_{31}$,

and so on. A discussion and analysis of ordering are found in Macklem [17].

We now investigate the relationship between C and $\nabla^2 f(x)$. The search directions are the orthogonal directions q_1, \dots, q_n and assume that the elements $(C_Q)_{ij}$ of the symmetric $m \times m$ matrix C_Q have been computed at the points

$$\{x^{ij}, x^{ij} + h^{ij}q_i, x^{ij} + k^{ij}q_j, x^{ij} + h^{ij}q_i + k^{ij}q_j\}, \quad (6)$$

for all (i, j) , $i \geq j$ and $(C_Q)_{ji}$ set to be equal to $(C_Q)_{ij}$. Let N be the union of all such points and let

$$\delta = \max_{z, y \in N} \|z - y\|, \quad (7)$$

and

$$\mathcal{N} = \left\{ x \in \mathbb{R}^n \mid \max_{y \in N} \|x - y\| \leq \delta \right\}. \quad (8)$$

Lemma 2: Assume that f is twice continuously differentiable and $\nabla^2 f$ is Lipschitz-continuous in \mathcal{N}

$$\|\nabla^2 f(x) - \nabla^2 f(y)\| \leq L\|x - y\|, \text{ for all } x, y \in \mathcal{N}.$$

Let the $n \times n$ symmetric matrix C_Q be computed with the points (6). Then any $x \in \mathcal{N}$ satisfies

$$\|QC_QQ^T - \nabla^2 f(x)\| \leq nL\delta. \quad (9)$$

The proof can be found in [6] and we will in the next section prove a more general result. In case of a quadratic function $L = 0$, the exact Hessian matrix is recovered. The second derivative is only required to be locally Lipschitz with respect to \mathcal{N} .

III. EXTENSION TO SEPARABLE FUNCTIONS

Suppose the function f is partially separable. As mentioned in the introduction, the Hessian will be sparse if f is twice continuously differentiable, and if the Hessian is not defined, the matrix of average curvature information will be sparse [7]. Let r be the number of nonzero elements in the lower diagonal of these curvature matrices. Then, even though the matrix C can be restricted to have this sparsity pattern, the matrix C_Q cannot be assumed to be sparse, since we cannot expect the finite differences (4) to be zero for arbitrary search directions \mathcal{Q} . However, sparsity can still be exploited.

Given two matrices $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{r \times s}$, the Kronecker product $A \otimes B$ is a $mr \times ns$ block matrix given as

$$A \otimes B = \begin{bmatrix} A_{11}B & \cdots & A_{1n}B \\ \vdots & & \vdots \\ A_{m1}B & \cdots & A_{mn}B \end{bmatrix}. \quad (10)$$

The Kronecker product is useful in the present context because of the relation

$$AXB = C \Leftrightarrow (B^T \otimes A)\text{vec}(X) = \text{vec}(C). \quad (11)$$

Here $\text{vec}(X)$ and $\text{vec}(C)$ are vectors containing the entries of the matrices X and C stacked row-wise [13].

Using (10) and (11) the rotation (5) can be written implicitly as

$$(Q^T \otimes Q^T)\text{vec}(C) = \text{vec}(C_Q). \quad (12)$$

Since we impose a sparsity structure on C as well as symmetry, all the entries in the upper triangle, as well as all the zero entries of $\text{vec}(C)$ can be removed from (12), resulting in the overdetermined equation system

$$(Q^T \otimes Q^T)P_c\overline{\text{vec}}(C) = \text{vec}(C_Q), \quad (13)$$

where the vector $\overline{\text{vec}}(C)$ contains the r elements of C to be determined, and the $n^2 \times r$ 0-1 matrix P_c adds together the columns corresponding to upper and lower diagonal elements C_{ij} and C_{ji} for all off-diagonal elements, and deletes the columns corresponding to zero entries in C . For example, if C is to be tridiagonal and is of size 3×3 , that is,

$$C = \begin{bmatrix} \times & \times & \\ \times & \times & \times \\ & \times & \times \end{bmatrix},$$

then it has one zero element and five nonzero elements in the lower triangle, so that P_c has size 9×5 and reads:

$$P_c = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \quad (14)$$

Since the equation system (13) is overdetermined, we can select r rows from the coefficient matrix and the right-hand side, resulting in the $r \times r$ equation system

$$\text{Prow}(Q^T \otimes Q^T)P_c\overline{\text{vec}}(C) = \text{Prow}\text{vec}(C_Q), \quad (15)$$

where Prow is an $r \times n^2$ 0-1 matrix which selects r rows. Prow will be the first r rows of a permuted $n^2 \times n^2$ identity matrix. The resulting equation system (15) will be significantly smaller than its counterpart (12) when a sparsity structure is imposed on C , and the corresponding effort required to compute the right-hand side is similarly smaller. If there are only $O(n)$ elements to be determined, then the number of steps needed to compute the entire right-hand side $\text{Prow}\text{vec}(C_Q)$ does not depend on n , which does away with the practical limit on dimension discussed in the previous section.

Exactly which rows Prow should select in order to create a well-conditioned coefficient matrix is nontrivial, and is sometimes called the subset selection problem in the literature (see e.g., [9]). One suitable solution procedure is to

determine these rows by computing a strong rank-revealing QR factorization of the transpose of $P_{\text{row}}(Q^T \otimes Q^T)$ and selecting the rows chosen by the theory and algorithms of Gu and Eisenstat, presented in [11]. An implementation of this selection procedure can be found in [21].

IV. CONVERGENCE THEORY

The method presented so far, being a sufficient decrease method with $2n$ search directions which are the positive and negative of n mutually orthogonal directions, adheres to the algorithmic framework and convergence theory of Lucidi and Sciandrone [15]. We can therefore state the following theorem, without proof.

Theorem 3: Suppose f is continuously differentiable, bounded below and the level set $\mathcal{L}(x) = \{y \mid f(y) \leq f(x)\}$ is compact. Then, the method converges to a first-order stationary point.

We now prove that if f is twice continuously differentiable, then the computed curvature matrix C converges to the true Hessian in the limit.

Define

$$A = P_{\text{row}}(Q^T \otimes Q^T)P_c.$$

Let f be twice continuously differentiable and Hessian Lipschitz-continuous in the sense that

$$\|\nabla^2 f(x) - \nabla^2 f(y)\| \leq L\|x - y\|. \quad (16)$$

Define r pairs of vectors $p^{(k)}, q^{(k)}$ $k = 1, \dots, r$, all of unit length, such that the k th row of A is equal to

$$\left(p^{(k)T} \otimes q^{(k)T} \right) P_c. \quad (17)$$

This means some of these vectors will be equal, but the pairs will be unique. In addition let r points x^k , $k = 1, \dots, r$, be such that element k of $P_{\text{row}}\text{vec}(C_Q)$,

$$(P_{\text{row}}\text{vec}(C_Q))_k = p^{(k)T} \nabla^2 f(x^k) q^{(k)}.$$

Let η be such that

$$\max_{i,j} \|x^i - x^j\| = \eta.$$

Let \mathcal{N} be the neighborhood of points such that

$$\mathcal{N} = \{x \mid \|x - x^k\| \leq \eta, k = 1, \dots, r\}.$$

For convenience, let us restate (15), as

$$\text{Avec}(C) = P_{\text{row}}\text{vec}(C_Q). \quad (18)$$

Lemma 4: Assume A is invertible. Let C be the symmetric $n \times n$ matrix constructed from the solution of (18). Then, there exists an $x \in \mathcal{N}$ such that

$$\|\nabla^2 f(x) - C\| \leq \|A^{-1}\| n L \eta.$$

Proof. Let us rewrite the contents of $P_{\text{row}}\text{vec}(C_Q)$:

$$\begin{aligned} & (P_{\text{row}}\text{vec}(C_Q))_k \\ &= p^{(k)T} \nabla^2 f(x^k) q^{(k)} \\ &= p^{(k)T} (\nabla^2 f(x) + \nabla^2 f(x^k) - \nabla^2 f(x)) q^{(k)} \\ &= \left[p^{(k)T} \nabla^2 f(x) q^{(k)} \right] + \\ &\quad \left[p^{(k)T} (\nabla^2 f(x^k) - \nabla^2 f(x)) q^{(k)} \right]. \end{aligned} \quad (19)$$

Then, and in addition defining $h = \overline{\text{vec}}(\nabla^2 f(x))$, equation (18) can be written as

$$A \overline{\text{vec}}(C) = Ah + \epsilon. \quad (20)$$

Here $(Ah)_k$ is the expression in the first parenthesis of (19), and ϵ_k is the expression in the last parenthesis of (19). If we consider the norm of a single element in ϵ , this is

$$\begin{aligned} |\epsilon_k| &\leq \|p^{(k)}\| \|\nabla^2 f(x^k) - \nabla^2 f(x)\| \|q^{(k)}\| \\ &\leq L\eta, \end{aligned} \quad (21)$$

using (16) and the fact that p and q have unit length. When solving (18), we get

$$\overline{\text{vec}}(C) = h + A^{-1}\epsilon.$$

If we consider a single element of $\overline{\text{vec}}(C)$ and h we can write

$$|(\overline{\text{vec}}(C))_k - h_k| \leq \|A^{-1}\| |\epsilon_k|,$$

which can also be written

$$|C_{ij} - (\nabla^2 f(x))_{ij}| \leq \|A^{-1}\| |\epsilon_k| \quad (22)$$

Using the property of the 2-norm that

$$\|A\|_2 \leq n \max_{i,j} |a_{ij}|,$$

as well as (21) we can extend (22) to

$$\|C - \nabla^2 f(x)\| \leq \|A^{-1}\| n L \eta,$$

which completes the proof. \square

We must now prove that there always exists a matrix A with rank r , and that the term $\|A^{-1}\|$ is uniformly bounded. Since A is made up of the rows of the matrix $(Q^T \otimes Q^T)P_c$, there will be a choice of rows which imply full rank if the matrix $(Q^T \otimes Q^T)P_c$ has rank r .

Lemma 5: For any orthogonal matrix Q and any sparsity structure to be imposed on C , the matrix $(Q^T \otimes Q^T)P_c$ has full rank r , and its smallest singular value σ_r satisfies $\sigma_r \geq 1$.

Proof. Since Q is orthogonal, so is Q^T , and also $(Q^T \otimes Q^T)$. For any sparsity structure, right-multiplying $(Q^T \otimes Q^T)$ with P_c either adds together two columns, or deletes columns. Consequently, the columns of the resulting matrix $(Q^T \otimes Q^T)P_c$ are orthogonal (which implies full rank), and have either length one or length $\sqrt{2}$. It then

follows that the singular values are equal to the length of the column vectors, either 1 or $\sqrt{2}$. \square

If we consider (14) and the corresponding $(Q^T \otimes Q^T)P_c$, the norm of first column of $(Q^T \otimes Q^T)P_c$ is 1 and the norm of the second column is $\sqrt{2}$.

Lemma 6: P_{row} can be chosen such that for a given n , the smallest singular value of A is uniformly bounded below, and consequently that $\|A^{-1}\|$ is uniformly bounded.

Proof. This result follows from the theory and methods of Gu and Eisenstat [11], which guarantee that the rows of A (or equivalently the columns of A^T , as is done in [11]) can be selected from the rows of $(Q^T \otimes Q^T)P_c$ in such a way that the smallest singular value of A is larger than or equal to the smallest singular value of $(Q^T \otimes Q^T)P_c$, divided by a low order polynomial in n and r . Since n and r are given and the smallest singular value of $(Q^T \otimes Q^T)P_c$ is always larger than or equal to 1, the result follows. \square

Finally, we show that η goes to zero as the GSS method converges to a stationary point.

Lemma 7: Assume that the step length expansion factor is uniformly bounded by, say, M . Then, as the step length δ go to zero, so does η .

Proof. That the step length δ goes to zero is an integral part of the convergence theory of GSS methods and is proved in e.g. [14]. η is the diameter of neighborhood of points \mathcal{N} . Since all the points in \mathcal{N} lie within the rectangles of points used in the formula (4), it follows that η must be smaller than maximum possible distance between the first and the last points used for computing C (the corner points of the rectangle $abcd$ in Figure 2). Suppose, that when the computation of C is started the step length is δ_{\max} , and that the maximum possible number of step length increases before C is computed is t . Then we have

$$\eta \leq \sum_{k=0}^t \delta_{\max} M^{k-1}.$$

The only variable in this expression is δ_{\max} , and we know it goes to zero as the method converges. Consequently, so must η . \square

This allows us to state the following theorem:

Theorem 8: Assume that f is twice continuously differentiable, bounded below and that the level sets $\mathcal{L}(x)$ are compact. Then, as the method converges, C converges to the true Hessian.

The proof follows from the preceding Lemmas. This result, together with the preliminary numerical results in this paper allows us to conjecture that the method actually converges to second-order stationary points.

V. TESTING DERIVATIVE FREE OPTIMIZATION METHODS

The purpose of the following sections is to report on numerical experiments on two unconstrained optimization problems where methods risk terminating at a saddle point,

avoiding a nearby strict local minimizer. For visualization purposes we have chosen problems with two unknowns.

The first problem is a modification of a problem suggested by Wolfe [26]. In its unmodified form this problem has been used to show that gradient based methods tend to converge to a saddle point. The modification will make the function bounded below and introduce a local minimizer but not change the region where gradient based methods converge to the saddle-point. The second example is a modification of a function presented in [1], which has a very narrow cone of negative curvature. Again the modification will make the function bounded below and introduce local minimizers.

Generating set search methods described in the previous sections are shown to converge to stationary points and the set of search directions in the limit will be the eigenvalues of the Hessian matrix at the solution. If the Hessian matrix at the stationary point has a negative eigenvalue, one of the search directions will be a descent direction. A generating set search should therefore not experience convergence to the saddle points of the two test functions. A generating set search method is compared with two methods which do not have the same property to generate descent at a saddle point.

In the second part of the experiments we compare the efficiency of GSS-CI with two classical derivative free methods on a class of test problems.

A. The methods

The three methods primarily used in testing, are GSS-CI, NEWUOA and NMSMAX. We will briefly discuss two additional methods, MDSMAX and fminsearch.

1) *GSS-CI:* This is the method presented in the previous sections and [8], [2], and is based on the method of [6]. Since the method gathers average slope information the method can consequently perform Newton-like steps at regular intervals.

The initial search directions are chosen to be the positive and negative coordinate vectors. Each pair of search directions (e.g. $\pm q_i$, where q_i is a search direction) has a step length δ_i associated with it. In our experiments these are initially set to the same value, $0.2\|x_0\|_1$, but they will be increased or decreased individually depending on the success or failure of the search along the corresponding pairs of search directions. A search is deemed successful if it produces sufficient decrease, that is, if

$$f(x + \delta_i q_i) < f(x) - \rho(\delta_i),$$

where $\rho : \mathbb{R} \mapsto \mathbb{R}$ is nondecreasing function satisfying a few technical requirements, outlined in [14]. In our implementation we use

$$\rho(\delta) = 10^{-4}\delta^2.$$

The termination criterion is that the product of all the step lengths should be less than or equal to a tolerance. In our

experiments this is

$$\prod_{i=1}^n \delta_i \leq \left(10^{-4} \|x_0\|_1\right)^n.$$

B. NEWUOA

NEWUOA [22] is an interpolation method, where the number of interpolation points can be determined by the user. The remaining degrees of freedom are taken up by minimizing the Frobenius norm of the difference between one Hessian approximation and the next.

An initial vector $x_0 \in \mathbb{R}^n$, the number m of interpolation points, and the initial and final values of a trust region radius, namely ρ_{beg} and ρ_{end} must be provided by the user. The number of interpolation points m is a fixed integer from the interval $n+2 \leq m \leq \frac{1}{2}(n+1)(n+2)$. It is recommended to use $m = 2n+1$ for efficiency. The initial interpolation points $x_i, i = 0, 1, 2, \dots, m$, have the property that $\|x_i - x_0\|_2 = \rho_{\text{beg}}, i = 1, 2, \dots, m$, unless $m > 2n + 1$, in which case this distance is $\sqrt{2}\rho_{\text{beg}}$. The termination criterion is related to the radius ρ_{end} .

C. NMSMAX

NMSMAX [12] is an implementation by Nicholas J. Higham of the classical Nelder-Mead simplex method [20]. The user can choose whether the initial simplex is right-angled or regular (with sides of equal length). The initial simplex size is not input by the user, but taken to be the order of $\max(\|x_0\|_\infty, 1)$. The method terminates when either the maximum number of function evaluations is reached, or when the relative size of the simplex, is below a certain threshold. That is,

$$\frac{1}{\max(1, \|v_0\|_1)} \max_{1 \leq i \leq n} \|v_i - v_0\|_1 \leq \text{tol}.$$

Here v_0 and $v_i, i = 1, \dots, n$ are the vertices of the simplex. In our experiments we use $\text{tol} = 10^{-6} \|x_0\|_1$.

D. The methods MDSMAX and fminsearch

We also included a brief test of the methods MDSMAX, which is an implementation by Nicholas J. Higham of the multidirectional search method due to Virginia Torczon [25], and fminsearch [18], which is the Matlab implementation of the Nelder-Mead method.

VI. THE FUNCTIONS

The two functions are in two variables, are twice continuously differentiable and bounded below.

A. Function I – A Narrow Positive Cone

The function (23) is a modification of a test function in [1]:

$$f(x, y) = (9x - y)(11x - y) + \frac{x^4}{2}. \quad (23)$$

It has a saddle point at the origin, and two local minimizers at $(x, y) = \pm(1, 10)$. Level curves for this function can be seen in Figure 8.

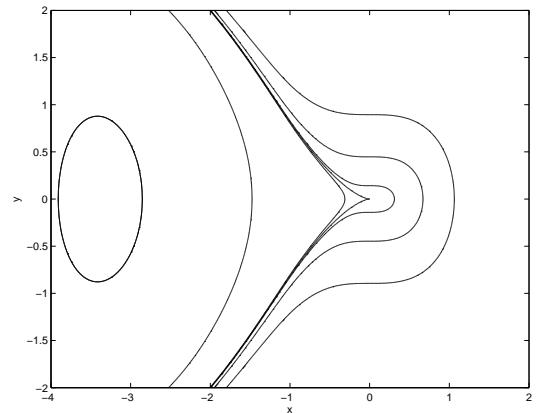


Figure 3. Level curves for the function (24).

B. Function II – Modified Wolfe Function

The second function (24) is a modified version of a test function due to Wolfe [26]:

$$f(x, y) = \frac{x^3}{3} + \frac{y^2}{2} - \frac{2}{3}(\min[x, -1] + 1)^3, \quad (24)$$

It has a saddle point at the origin and a minimum at $(x, y) = (-2 - \sqrt{2}, 0)$. Level curves for this function are shown in Figure 3. The original function (not bounded below) is given by $f(x, y) = \frac{x^3}{3} + \frac{y^2}{2}$.

VII. NUMERICAL EXPERIENCE

Region of Convergence: The region of convergence of a stationary point is the set of starting points for which a given method terminates close to the stationary point. In addition to the input parameters for the methods we need to specify the tolerance (or distance between) the terminating point and the stationary point. A globally convergent method on a sufficiently smooth function is characterized by for all starting points, the method will for any $\varepsilon > 0$ generate an iterate x_k so that $\|\nabla f(x_k)\| \leq \varepsilon$. However, the stopping criteria of the implementation may be based on changes in the function values or on the difference between two iterates. Even the case $\|\nabla f(x_k)\| \leq \varepsilon$ will in general not guarantee that the distance between the stationary point and x_k is small. We can thus expect that even if the methods terminate successfully, the distance to a stationary point will not be smaller than the tolerance for some starting points. For simplicity we say that a method terminates at a stationary point when it terminates at a point that satisfies the tolerance.

A. Function I

For this function we generate starting points in the fourth quadrant $\{(x, y) | x \leq 0, y \geq 0\}$. The minimizers of the function are in the first and third quadrants, so we expect the methods to terminate successfully at the minimizers

if started in these quadrants. This is confirmed in the preliminary numerical testing. Furthermore, because of the symmetry of the function we can choose either the second or fourth quadrants, at least for GSS-CI and NEWUOA.

GSS-CI: We discretize the area $[-8, 0] \times [0, 10]$, into a 201×201 grid of points, and start the method with initial step length $0.2\|x_0\|_1$ for all directions, and the termination criterion is that the product of all the step lengths should be less than or equal to $(10^{-4}\|x_0\|_1)^n$. (If $x_0 = 0$ then nonzero values are used.) The results are given in Figure 4. In the figure, a blue color means that the method terminated

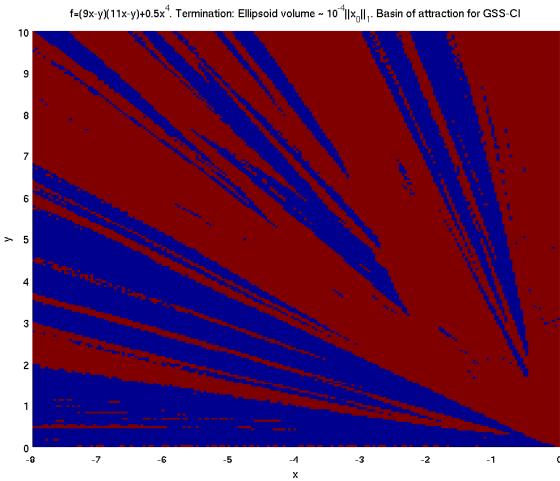


Figure 4. GSS-CI on the function $f = (9x_1 - x_2)(11x_1 - x_2) + \frac{1}{2}x_1^4$. The initial step length is $0.2\|x_0\|_1$ for all directions, and the termination criterion that the volume of the ellipsoid defined by the scaled search directions should be proportional to $10^{-4}\|x_0\|_1$, that is, $\prod_i \delta_i \leq [10^{-4}\|x_0\|_1]^n$. (The discretization is 201×201 .)

close to $(x, y) = (1, 10)$ for the corresponding starting point, red color means termination close to $(x, y) = (-1, -10)$. As one can see, the method does not terminate close to the saddle point for any of these starting points. When starting at the origin, setting a nonzero step length results in convergence to a minimizer.

NEWUOA: We generate starting points on a 1001×1001 discretization of the region $[-8, 0] \times [0, 10]$, and run NEWUOA with parameters $\rho_{\text{beg}} = 0.2\|x_0\|_1$, and $\rho_{\text{end}} = 10^{-5}\|x_0\|_1$. (Once again, if $x_0 = 0$ then nonzero values are used.) The results are visualized in Figure 5. As before, blue color means that the method terminated close to $(x, y) = (1, 10)$ for the corresponding starting point, red color means termination close to $(x, y) = (-1, -10)$. In addition, green means termination close to the origin, and orange means none of the above. As one can see, the method does terminate close to the saddle point for some starting points, and these points make up a small region on the border between the basins of attraction of $(x, y) = (1, 10)$ and $(x, y) = (-1, -10)$.

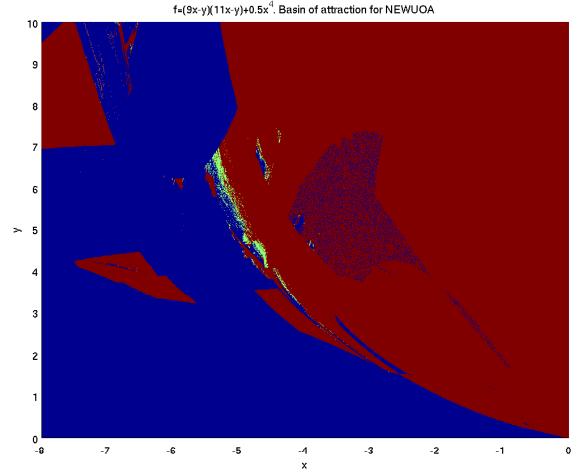


Figure 5. Plot of basins of attraction for NEWUOA on $f = (9x_1 - x_2)(11x_1 - x_2) + \frac{1}{2}x_1^4$, with $\rho_{\text{beg}} = 0.2\|x_0\|_1$ and $\rho_{\text{end}} = 10^{-5}\|x_0\|_1$. (The discretization is 1001×1001 .)

To check if the basins of attraction are sensitive to the termination criterion we repeat the experiment, but this time with $\rho_{\text{end}} = 10^{-6}\|x_0\|_1$. The results are given in Figure 6. As we can see in this figure, the starting points for which the method terminates at the saddle point are still wedged between the red and blue regions, but the green region is now much smaller.

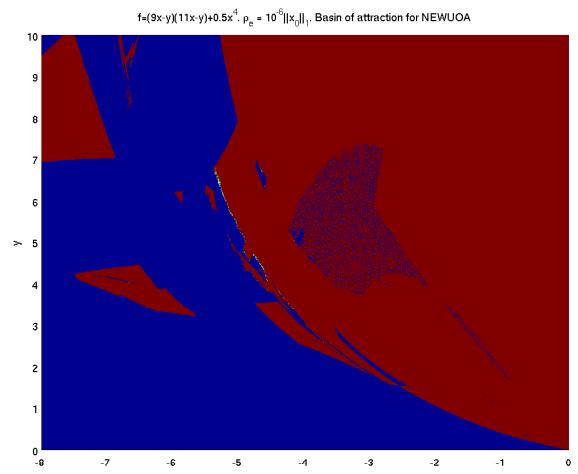


Figure 6. Decreasing ρ_{end} in NEWUOA to $10^{-6}\|x_0\|_1$ will basically not change the region of convergence for the local minimizers, but the region of convergence to the saddle-point gets smaller, squeezed between the regions of convergence to the local minimizers. (The discretization is 1001×1001 .)

Similarly, we test what happens with a looser convergence criterion, namely $\rho_{\text{end}} = 10^{-4}\|x_0\|_1$. The results are in

Figure 7. For this convergence criterion the green region is

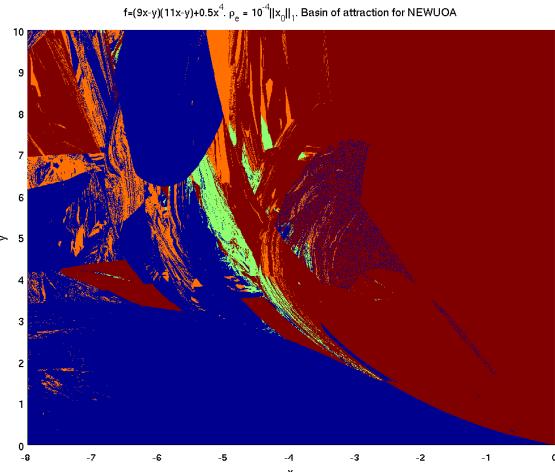


Figure 7. Increasing ρ_{end} in NEWUOA to $10^{-4} \|x_0\|_1$ will force many starting points not to be accepted as close to a stationary point. The regions are basically the same, but the region of convergence for the saddle-point is larger. (The discretization is 1001×1001 .)

much larger, and there are also large orange regions, which correspond to termination no closer to any of the stationary points than 0.2.

NMSMAX: For NMSMAX, we discretize the region $[-10, 10] \times [-10, 10]$ into a 201×201 grid. We choose a right-angled initial simplex. The size of the initial simplex is not determined by the user, but we set the termination criterion to be a simplex size of $10^{-6} \|x_0\|_1$. The results, as well as level curves of the function are given in Figure 8. As one can see, for about half the fourth quadrant the method terminates at the saddle point, even though the convergence criterion is quite strict. In addition, termination at the saddle point occurs for points close to the negative y -axis, and along

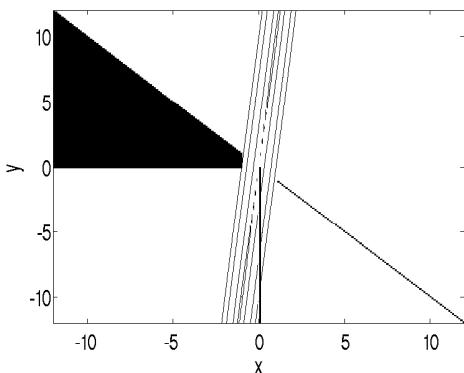


Figure 8. Level curves of the function (23), and starting points for which NMSMAX terminates at the saddle point at the origin, marked in black.

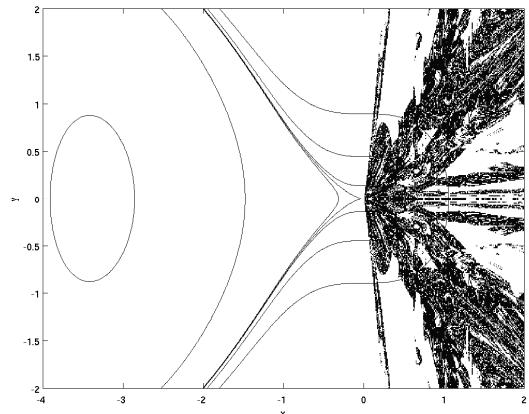


Figure 9. Level curves for the function (24) as well as starting points for which NEWUOA terminates at the saddle point at the origin, marked in black.

the line $y = -x$.

It is also interesting to note that in this case, the behavior in quadrants two and four are *not* the same.

B. Function II

For this function we discretize the region $[-4, 2] \times [-2, 2]$ into a 601×401 grid.

GSS-CI: Using the same parameter settings as for function I, GSS-CI once again terminates at the (single) minimizer, so an attraction basin plot would simply be the region filled with one color. (When $x_0 = 0$, nonzero step lengths are used, and the method converges to the minimizer.)

NEWUOA: For this function we also use the same parameter values as before, namely $\rho_{\text{beg}} = 0.2 \|x_0\|_1$ and $\rho_{\text{end}} = 10^{-6} \|x_0\|_1$. The results are in Figure 9. As one can see, there is a relatively large collection of points in the first and second quadrants, for which the method terminates at the saddle point at the origin.

To see if the cause of this behavior was the number of interpolation points ($2n + 1$ in this case), we also tried a full quadratic model, by using six interpolation points. The results for this case are in Figure 10. As one can see, the black region now has a different shape, but is located approximately in the same position, and is of similar size.

NMSMAX: For this function, NMSMAX terminates at the saddle point for a few starting points on the y -axis only.

VIII. TESTING THE METHODS MDSMAX AND FMINSEARCH

MDSMAX: The results are reported in Figures 11 and 12 and Figures 13 and 14. For the function (24) termination close to the saddle points rarely occurs, and when it does the corresponding starting points lie along straight lines, one at the upper right corner of Figure 14, and one on the y -axis close to the bottom of the figure using right angled simplex.

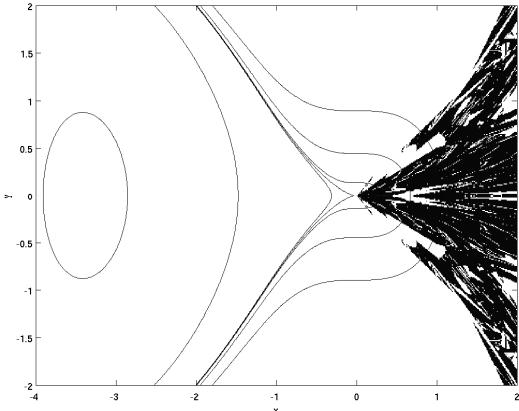


Figure 10. Level curves for the function (24) as well as starting points for which NEWUOA terminates at the saddle point at the origin, marked in black. Number of interpolation points $m = 6$.

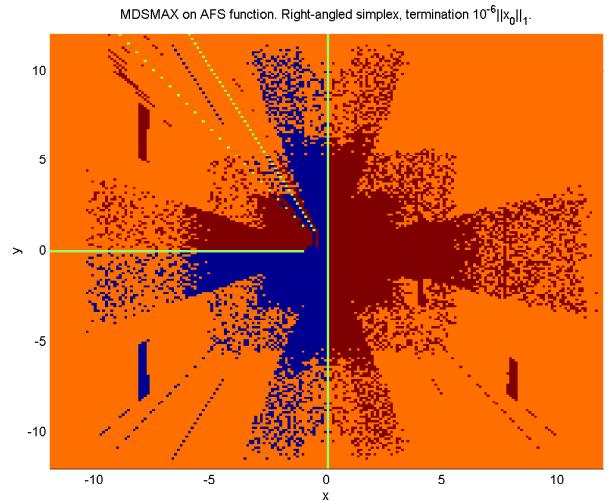


Figure 12. MDSMAX on the function $f = (9x - y)(11x - y) + \frac{x^4}{2}$. Red means termination close to $(x, y) = (1, 10)$, blue means termination close to $(x, y) = (-1, -10)$, green termination close to the saddle point at the origin, and orange means stagnation. Right-angled simplex.

However, MDSMAX has serious problems with stagnation on the function (23), as can be seen in Figures 11 and 12.

fminsearch: This method has few problems on these functions, except when the starting points lie on one of the axes. For the function (23), 208 of the 40401 starting points result in termination close to the saddle point, 201 of these 208 points being on the x -axis. For the function (24), 890 of the 241001 starting points result in termination close to the saddle point, all of these on or immediately next to the y -axis. These results were obtained using the standard convergence tolerances. Tightening the convergence criteria gives an even more favorable result.

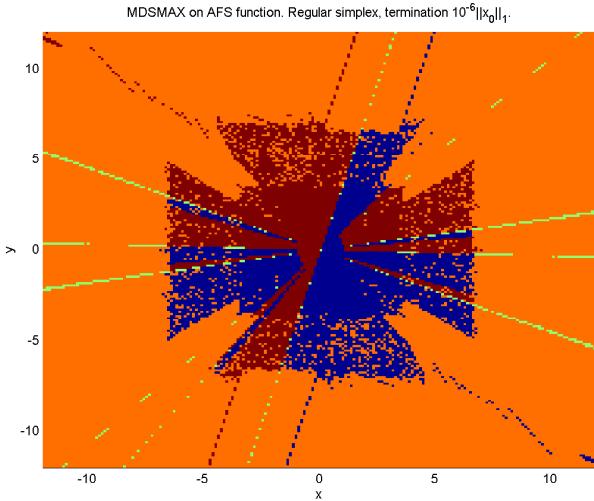


Figure 11. MDSMAX on the function $f = (9x - y)(11x - y) + \frac{x^4}{2}$. Red means termination close to $(x, y) = (1, 10)$, blue means termination close to $(x, y) = (-1, -10)$, green termination close to the saddle point at the origin, and orange means stagnation. Regular simplex.

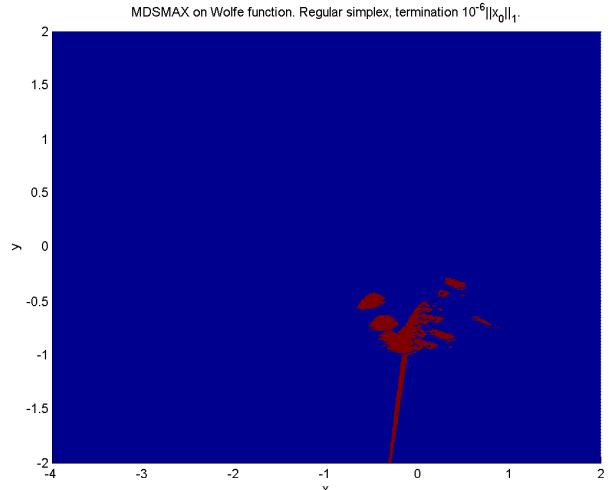


Figure 13. MDSMAX on the function $f = \frac{x^3}{3} + \frac{y^2}{2} - \frac{2}{3}(\min[x, -1] + 1)^3$. Blue means termination close to the minimum at $(x, y) = (-2 - \sqrt{2}, 0)$, red means termination close to the saddle point at the origin. Regular simplex.

IX. EFFICIENCY

Moré and Wild [19], benchmarked different derivative-free optimization solvers on 53 smooth problems. In this test we use the same set of problems and two of the same solvers (NEWUOA and NMSMAX) as Moré and Wild [19]. We run the three methods on each problem, and declare a success if a method uses less than 5000 function evaluations, and the gradient corresponding to the solution satisfies $\|\nabla f(x)\| \leq 10^{-2}$, where this gradient is computed with finite differences. The corresponding data profile is

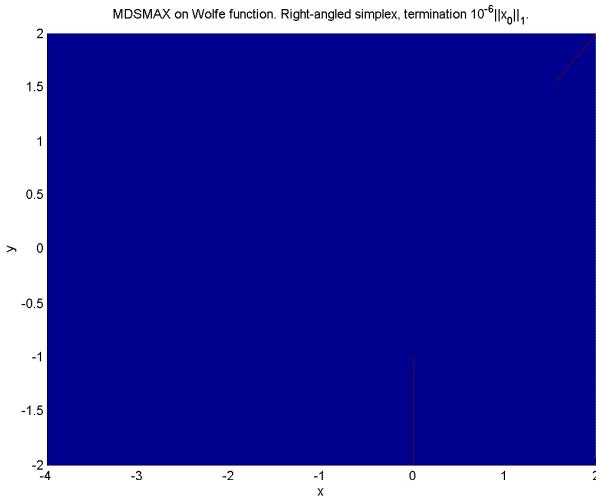


Figure 14. MDSMAX on the function $f = \frac{x^3}{3} + \frac{y^2}{2} - \frac{2}{3}(\min[x, -1] + 1)^3$. Blue means termination close to the minimum at $(x, y) = (-2 - \sqrt{2}, 0)$, red means termination close to the saddle point at the origin. Right-angled simplex.

shown in Figure 15. The horizontal axis is number of equivalent gradient evaluations, i.e. one equivalent gradient is n function evaluations. As one can see from the figures NEWUOA solves the most problems if one has a tight computational budget, and GSS-CI solves the most problems if one has as moderate computational budget. NMSMAX performs the weakest among the methods on these functions.

X. DISCUSSION

The simplex method [20] is one of the most used derivative free optimization methods. In this note we have used the implementations NMSMAX and fminsearch of the simplex method. The other three methods discussed in the paper represents different approaches of derivative free optimization. We have shown that GSS-CI solves the most problems if one has as moderate computational budget compared to NEWUOA and NMSMAX. The methods NEWUOA, NMSMAX, and fminsearch may terminate close to the saddle point while GSS-CI will not converge to the saddle point for these two examples. This supports the observed convergence properties of GSS-CI. The regions of convergence are dependent on the input parameters and the results presented are typical behavior of the methods.

ACKNOWLEDGEMENTS

The authors would like to thank Mike Powell and Nicholas J. Higham for helpful comments on the experiments and on an earlier version of the section on basin of attraction. The authors would also like to thank Marielba Rojas for the help on rank revealing QR.

The first author gratefully acknowledges partial funding from The Norwegian Research Council, Gassco and Statoil.

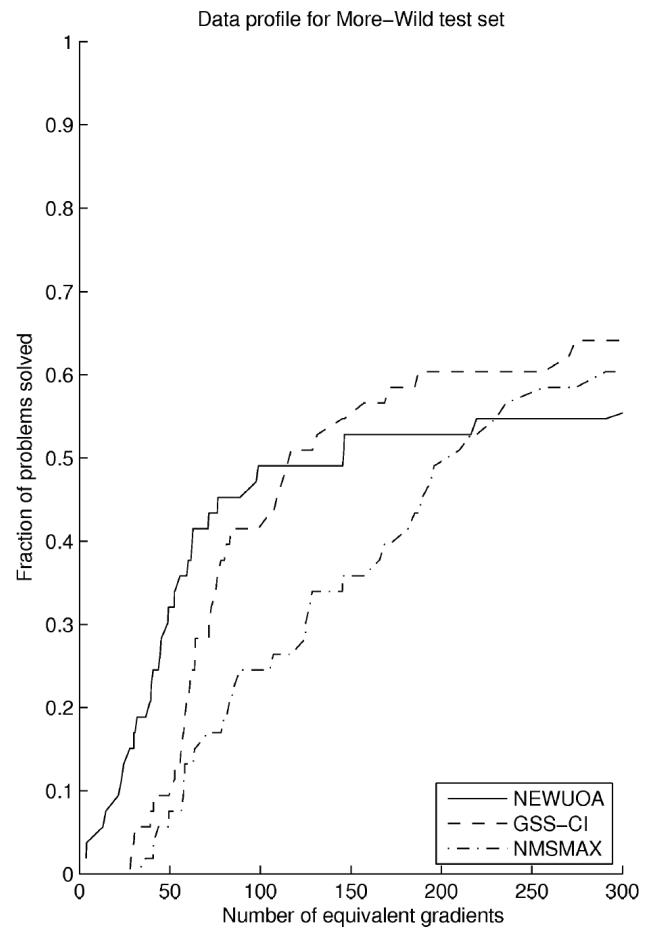


Figure 15. Data profile for the smooth functions of the test set of Moré and Wild [19].

REFERENCES

- [1] M. A. Abramson. Second-order behavior of pattern search. *SIAM Journal on Optimization*, 16(2):315–330, 2005.
- [2] M. A. Abramson, L. Frimannslund, and T. Steihaug. A subclass of generating set search with convergence to second-order stationary points. To be submitted, 2011.
- [3] A. R. Conn, K. Scheinberg, and L. N. Vicente. *Introduction to Derivative-Free Optimization*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2009.
- [4] I. D. Coope and C. J. Price. A direct search conjugate directions algorithm for unconstrained minimization. *The ANZIAM Journal*, 42(E):C478–C498, 2000.
- [5] C. H. Edwards. *Advanced Calculus of Several Variables*. Academic Press, 1973. ISBN 0-12-232550-8.
- [6] L. Frimannslund and T. Steihaug. A generating set search method using curvature information. *Computational Optimization and Applications*, 38(1):105–121, 2007.

- [7] L. Frimannslund and T. Steihaug. Sparsity of the average curvature information matrix. *PAMM, Proc. Appl. Math. Mech.*, 7:1062101–1062102, 2007.
- [8] L. Frimannslund and T. Steihaug. A new generating set search algorithm for partially separable functions. In *Proceedings ADVCOMP 2010: The Fourth International Conference on Advanced Engineering Computing and Applications in Sciences*, pages 65–70. The International Academy, Research and Industry Association (IARIA), 2010. ISBN:978-1-61208-000-0.
- [9] G. H. Golub and C. F. van Loan. *Matrix computations*. Johns Hopkins University Press, Baltimore, MD, USA, 1996.
- [10] A. Griewank and Ph. L. Toint. On the unconstrained optimization of partially separable functions. In M. Powell, editor, *Nonlinear Optimization 1981*, pages 301–312. Academic Press, 1982.
- [11] M. Gu and S. C. Eisenstat. Efficient algorithms for computing a strong rank-revealing QR factorization. *SIAM Journal on Scientific Computing*, 17(4):848–869, 1996.
- [12] N. J. Higham. The Matrix Computation Toolbox. <http://www.ma.man.ac.uk/~higham/mctoolbox>.
- [13] R. A. Horn and C. R. Johnson. *Topics in matrix analysis*. Cambridge University Press, Cambridge, United Kingdom, 1991.
- [14] T. G. Kolda, R. M. Lewis, and V. Torczon. Optimization by direct search: New perspectives on some classical and modern methods. *SIAM Review*, 45:385–482, 2003.
- [15] S. Lucidi and M. Sciandrone. On the global convergence of derivative-free methods for unconstrained optimization. *SIAM Journal on Optimization*, 13(1):97–116, 2002.
- [16] M. Machura and A. Mulawa. Algorithm 450 Rosenbrock function minimization. *Communications of the ACM*, 16(8):482–483, 1973.
- [17] M. Macklem. *Low Dimensional Curvature Methods in Derivative-free Optimization on Shared Computing Networks*. PhD thesis, Computer Science, Dalhousie University, Halifax, Nova Scotia, Canada, 2009.
- [18] The MathWorks, Inc., Natick, Massachusetts, USA. *MATLAB Optimization Toolbox User's Guide*, 2010.
- [19] J. J. Moré and S. M. Wild. Benchmarking derivative-free optimization algorithms. *SIAM Journal on Optimization*, 20(1):172–191, 2009.
- [20] J. A. Nelder and R. Mead. A simplex method for function minimization. *The Computer Journal*, 7:308–313, 1965.
- [21] S. R. Pope. *Parameter Identification in Lumped Compartment Cardiorespiratory Models*. PhD thesis, North Carolina State University, Raleigh, North Carolina, USA, 2009.
- [22] M. J. D. Powell. *Large-Scale nonlinear optimization*, volume 83 of *Nonconvex Optimization and its applications*, chapter The NEWUOA software for unconstrained optimization without derivatives, pages 255–297. Springer US, 2006.
- [23] C. P. Price and Ph. L. Toint. Exploiting problem structure in pattern search methods for unconstrained optimization. *Optimization Methods and Software*, 21(3):479–491, 2006.
- [24] H. H. Rosenbrock. An automatic method for finding the greatest or least value of a function. *The Computer Journal*, 3(3):175–184, Oct. 1960.
- [25] V. Torczon. *Multi-Directional Search: A Direct Search Algorithm for Parallel Machines*. PhD thesis, Department of Mathematical Sciences, Rice University, Houston, Texas, 1989. Available as Tech. Rep. 90-07, Department of Computational and Applied Mathematics, Rice University, Houston, Texas 77005-1892.
- [26] P. Wolfe. Convergence conditions for ascent methods. II: Some corrections. *SIAM Review*, 13(2):185–188, 1971.

A Systematic Review and Taxonomy of Runtime Invariance in Software Behaviour

Teemu Kanstrén

VTT Technical Research Centre of Finland

Oulu, Finland

teemu.kanstren@vtt.fi

Abstract— Describing software runtime behaviour in terms of its invariant properties has gained increasing popularity and various tools and techniques to help in working with these invariants have been published. These typically take a specific view on the possible and supported properties. In many cases it is also useful to view these in a wider context to enable a deeper understanding of possible invariance and to provide more extensive support across different domains. This paper aims to identify different aspects of the runtime invariance based on a review of existing works, and to present these results in a taxonomy that positions the different aspects in relation to each other. The goal is to provide support for their use in practice and to help identify possible research directions. A systematic review has been performed to identify relevant works in the literature. From these, a set of relevant properties have been collected to form the taxonomy. The resulting taxonomy has been structured to describe the different properties of runtime invariance. One main axis gives an overview of usage domains. One describes process related properties that are further classified to specification and evaluation related properties. A third main axis describes properties of runtime invariance itself and is further classified to properties of measurements, patterns and scope. It is concluded that the taxonomy provides a representation of different properties of runtime invariance used in current works. It can be used as a basis for modelling and reasoning about software runtime behaviour generally or as a basis for specialization in different domains.

Keywords-systematic review; software behaviour; runtime invariance; taxonomy

I. INTRODUCTION

This paper extends on previous work presented in [1]. Runtime invariance as discussed in this paper describes software behaviour in terms of its invariant properties as observed through dynamic analysis. Dynamic analysis uses as its basis information captured as observations from a (finite) set of program executions, such as test executions [2]. In line with these definitions, runtime invariance is defined here similar to [3] as a set of properties that hold at a certain point or points in a program execution. As a distinction from some uses of the term “invariant properties”, in this paper runtime invariance includes not only separate program points but also invariants over the overall behaviour of the observed system. That is, runtime invariance in this context refers to properties that are true for every observed execution. Recently the use of such invariants has become an increasingly popular technique in supporting different software engineering tasks (e.g., [4,5,6]).

Examples of runtime invariance include data-flow constraints (e.g., x always greater than 0 [3]), control-flow constraints (e.g., request always followed by a reply [7]), or their combinations (e.g., x is always greater than 0 when request is followed by a reply [8]). Runtime invariance can be specified manually as a model of expected behaviour for further processing with automated tools (e.g., [7]) or built (mined) based on observed behaviour (e.g., [3]). A model based on observed behaviour can also be referred to as describing likely invariance as it is based on observations made from a set of program executions, which typically do not cover the entire program behaviour state-space [3].

The idea of documenting and using invariants to reason about program behaviour at run-time can be seen to be as old as programming itself ([9,10]). Using invariants expressed in first-order logic to capture formal constraints on program behaviour was introduced as early as 1960's [9] by the pioneering work of Floyd [11] and Hoare [12].

Runtime invariance can be used in a variety of software engineering tasks and domains, such as helping in program comprehension [3], behaviour enforcement [13], test generation and oracle automation [6], or debugging [14]. Thus, when explicitly defined, a set of runtime invariants forms a basis for building automated support for many domains of software engineering.

There exist a number of tools to support the use of runtime invariance in different tasks (e.g., [3,6,15]). Many of these tools use a specific set of invariants for a specific domain. When applying runtime invariance in different domains, it is useful to also consider them in a wider context. When a set of invariants needs to be provided, either as manually defined input for evaluation by an automated processing tool, or as output (templates) from an automated specification mining tool, being able to generally reason about this invariance is needed for their effective use.

This paper describes a taxonomy of runtime invariance in software behaviour, describing different properties of these invariants. This is based on review of existing works on research and use of such invariants. The study is structured to describe how the invariants are specified and used, what kind of invariant patterns over software behaviour they capture, in which scope of behaviour they apply, and what information about the system behaviour is needed to be able to express and evaluate them.

The goal is to provide a systematic definition of the different properties of runtime invariance in software behaviour based on existing work, to facilitate their use in practice and to help form a basis for identifying future research directions.

This paper is structured as follows. Section II describes the overall approach taken to perform the survey and to create the taxonomy. Section III presents the taxonomy itself. Section IV provides two examples of applying the taxonomy. Finally, Section V provides discussion followed by the concluding remarks.

II. TAXONOMY BUILDING APPROACH

The taxonomy presented in this paper is based on performing a review of existing works in use and research on runtime invariance of runtime software behaviour. This review follows guidelines for performing systematic literature reviews (SLR) from [16]. SLR has been shown to be a robust and reliable research method in software engineering research [17]. In relation to the most common reasons for performing a SLR defined in [16], the intent here is to summarize existing works related to the use and research on the different properties of runtime invariance in software behaviour. Additionally, while not directly addressed by the resulting taxonomy, also two other most common reasons for a SLR defined in [16] can be observed as being supported by the provided taxonomy. Regarding these two, the taxonomy can be used as a framework to help position new research activities in the area, and to help identify gaps in current research.

A SLR can be defined in terms of the following features: a review protocol, a search strategy, selection criteria, and the definition of the information to be obtained from each included study [16]. Finally, data also needs to be synthesized to summarize the results [16]. The rest of this section discusses the approach taken in more detail relative to each of these features. The approach for the survey and taxonomy creation is also inspired by the taxonomy building approaches taken by Ducasse et al. [18] and Kagdi et al. [19], as well as the approaches for SLR taken by Cornelissen et al. [2] and Ali et al. [20].

A. Development of a Review Protocol

As noted before, the approach taken in this paper follows the guidelines for performing a SLR given in [16]. According to this, a review protocol should specify the methods that are used to perform the SLR. This includes defining the research questions, the search strategy, selection criteria, data extraction method, and synthesis approach [16]. Figure 1 illustrates the development process for the review protocol taken in the study presented in this paper. The first step is identifying the research questions that the survey is aiming to answer. The second step defines the search scope in terms of resources (journals, conferences, etc.) to be searched and the strategy for searching these resources. The selection criteria define what studies will be included in the SLR. A separate step of quality assessment is also possible at this point but in this paper this is embedded in the selection criteria as will be discussed in the following subsections. The data extraction strategy defines how the relevant information from each chosen study is extracted. Finally, the data needs to be synthesized to answer the research questions.

Piloting the different steps is also important in order to be able to identify any mistakes and problems in the procedures.

For the study presented in this paper, a preliminary study was conducted and published as a conference paper [1]. For this pilot study, comments were requested from experts in the field and also received from the conference peer-review (identified in the acknowledgements). The feedback from these instances was incorporated into different phases of the review protocol, which was then applied to produce the extended version of the study presented in this paper.

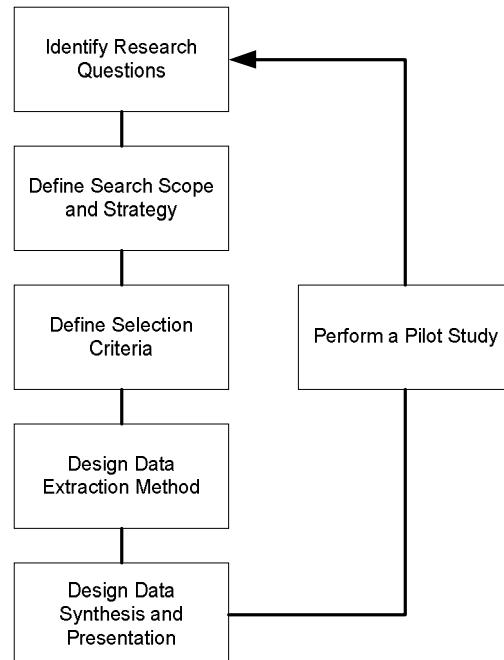


Figure 1. Development process for the review protocol.

B. Research Questions

The research questions should support the goal of the study, which here has been stated as providing a systematic definition of the different properties of runtime invariance in software behaviour in order to facilitate their use in practice and to help in identifying future research directions. To support this goal, the following research questions are addressed:

RQ-1: What are the properties of the processes used in analyzing software behaviour in terms of its runtime invariance?

RQ-2: What are the properties used to describe runtime invariance in software behaviour?

The first question is related to how the invariants over software runtime behaviour are used, and the second one to how the invariants themselves are defined.

C. Search Strategy

For reasons similar to those presented in [2], the main approach for performing the search has been manually over the selected publication venues. These reasons include the lack of support in current software engineering libraries for the identification of relevant research and primary studies, and the lack of common keywords across different venues or any such standard in relation to runtime invariance.

Table 1. Venues for article selection.

Abbr.	Description
ASE	International Conference on Automated Software Engineering
CSMR	European Conference on Software Maintenance and Reengineering
FSE	European Software Engineering Conference / Symposium on the Foundations of Software Engineering
ICSE	International Conference on Software Engineering
ICSM	International Conference on Software Maintenance
ICST	International Conference on Software Testing
ISSTA	International Symposium on Software Testing and Analysis
WCORE	Working Conference on Reverse Engineering
IST	Information and Software Technology
JSME	Journal of Software Maintenance and Evolution
JSS	Journal of Systems and Software
STVR	Software Testing, Verification and Reliability
TOSEM	ACM Transactions of Software Engineering and Methodology
TSE	IEEE Transactions on Software Engineering

The search venues are described in Table 1. The first eight of these are conferences and the last five are journals. The timeframe for the initial paper selection is from January 2001 until October 2010. This timeframe is chosen in order to provide a reasonable scope for the survey similar to those of [2] and [20]. It also scopes the start of the survey around the publication of one of the seminal papers on analysis of runtime invariance by Ernst et al. [3]. The selection of venues is based on the selection of well-known conferences and journals in the area of general software engineering and runtime analysis, similar to those in [2] and [20]. Additionally, it was scoped by the results of the pilot study ([1]), which started by examining the related publications collected over time on the Daikon tool website [21] (the tool originally described in [3]) and by performing keyword searches over digital library databases (e.g., IEEE Xplore, ACM Digital Library). The search venues in this paper are a composition from these different inputs, focusing on those that were found to be most relevant in the pilot study.

Similar to [2], in addition to the initial article selection from the venues listed in Table 1, interesting references from the chosen papers in the initial selection were also checked and relevant ones included in the survey. Where the authors' expertise in the field allowed to identify additional relevant references (e.g., [22]), these were also included.

As the focus of the study is on runtime invariance from the dynamic analysis viewpoint, the selection of papers and venues is also focused on the domain of analysing software runtime behaviour via dynamic analysis. However, as this can be seen to share many relevant properties with domains such as formal specification in general, also relevant works in other domains as referenced from the main body of works have been included in the taxonomy. However, to provide a

clear scope for the survey and to limit the scope of the study to a reasonable set, further exploration of the relations of the given taxonomy to other domains such as the formal methods community is left as a topic for future works.

D. Selection Criteria and Process

With regards to the research questions, two selection criteria for choosing the papers to be included were defined:

1. The selected papers directly discuss the use of invariants in relation to runtime software behaviour
2. The papers discuss modelling software behaviour in terms of properties that can be observed during runtime. When these models can be viewed in terms of invariance, they are considered relevant.

Table 2 lists the number of selected papers in relation to the selected venues. Due to the very large number of papers altogether (5817), it was not possible to read every paper fully. Instead, for each paper the title and abstract were first checked for relevance. If this provided no conclusion, the introduction and the conclusions were also reviewed for relevance. Finally, if needed, the full paper was read in order to define its suitability for inclusion. The total number of papers fully read is listed in the third column in Table 2 and is overall 348 papers.

The row titled "other" in Table 2 refers to papers that were included from venues other than the ones listed in Table 1. These come from the survey done in the pilot study, which included the papers listed on the Daikon website, digital library searches and from the reference checking in this extended study. Unfortunately the total number of such papers was not recorded during the pilot study and thus only the number of selected papers is given for these venues. The "selected" numbers in Table 2 reflect the references linked to the different axes of the taxonomy described in section III.

Table 2. Overview of study selection.

Venue	Papers	Read	Selected
ASE	406	49	12
CSMR	343	19	2
FSE	306	14	6
ICSE	481	25	20
ICSM	587	16	3
ICST	152	16	2
ISSTA	173	52	8
WCORE	277	17	3
IST	810	25	6
JSME	174	9	0
JSS	1270	29	3
STVR	101	11	2
TOSEM	132	8	3
TSE	605	58	7
Other	-	-	16
Total	5817	348	93

As mentioned, the process of SLR can also include a separate step of quality assessment after the paper selection step. Related to this, an exclusion criterion was also used. A paper was included only if it was observed as contributing

something new to the building of the taxonomy. For example, when a paper presents some further research on specific properties of invariance already discussed in a previous paper, only the earlier paper is included. This is an approach similar to that taken by Ducasse et al. [18], and does not mean that other papers would not be interesting. It simply scopes the study from the research question perspective.

Thus the taxonomy does not aim to include all possible papers dealing with runtime invariance but rather those in the selected venues that contribute to the definition of a taxonomy observed as best capable of answering the research questions.

E. Data Extraction and Synthesis

In building the taxonomy, an initial version of the main axes and their classes was defined based on the seminal work of Ernst et al. [3] on analyzing the runtime invariance of software behaviour. This was then refined based on review of other works and how these contributed to evolving the taxonomy and its different properties. This resulted in a more advanced and more fully structured version of the taxonomy. At this point the pilot study was submitted for conference peer-review. As noted before, based on the received feedback a more systematic survey was conducted in terms of a SLR, which was then used to update the taxonomy similar to the way the initial approach is described above.

In building the taxonomy, the Protégé ontology editor tool was used to capture and describe the different aspects and classes of the taxonomy. An adapted version of Binder's "fishbone" diagram [23] is used in section III to describe the different aspects of the taxonomy, focusing on specific properties one at a time. For space reasons, the description of the different properties is kept at a general level and for specific (and possibly more formal) definitions the reader is referred to the original references given.

The descriptions of the fishbone show the high-level properties as **bold**, mid-level properties as ***italics bold***, and specific properties in *italics*. As noted before, given references are not intended to be all-inclusive but to give a set of examples. However, for clarity and space no "e.g." is repeated for all of the references.

III. TAXONOMY OF RUNTIME INVARIANCE

This section presents the actual taxonomy of runtime invariance in software behaviour created based on the performed literature review. It is split into four subsections. The first subsection defines the main axes of the taxonomy and shows the overall picture. The second subsection describes different usage domains for such invariants. The third subsection describes the properties of the invariants themselves. Finally, the fourth subsection describes process-related properties for how such invariants are specified and evaluated. As noted before, the intent is to provide coverage of the different elements while including references observed as adding new elements to the taxonomy. Thus the intent is not to provide full coverage of all possible references for the described properties.

A. Main Axes

The generic process flow of using invariants in analysing software runtime behaviour, along with related properties for each step, is presented in Figure 2. This flow can be described as starting with *specification* of the *invariant information* that describes the software runtime behaviour of interest. Analysing the runtime behaviour requires capturing a set of observed measurements as a basis for the analysis, termed here as *measurement*. Depending on how the specification is done (automated mining vs. manual specification), it can also be interlinked with the measurement phase. Finally, in the *evaluation* phase the specified model of expected runtime invariance is compared against the observed model of actual runtime invariance.

The specification step is influenced by a set of *specification properties* that describe how the *invariant information* is formed. The invariant information describes the expected invariance and is formed as output from this step. This is then used as input for the measurement and evaluation steps. All steps in this process are related to the *usage domain* of the process. This means that the different phases of the process are impacted by the intended usage domain. The step of evaluation itself is described in this paper in terms of *evaluation properties*, which describes the general domain-independent properties of evaluation.

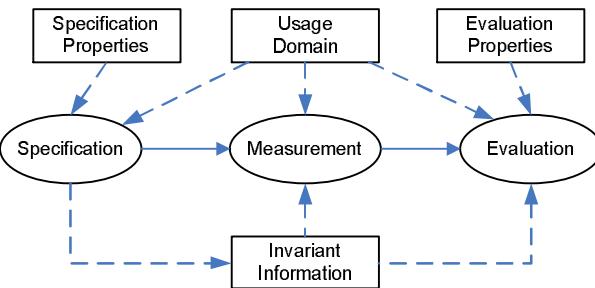


Figure 2. Flow of Elements.

The main axes of the taxonomy are divided into one axis describing the usage domains of runtime invariance, two for describing the process-related properties and three axes related to properties of runtime invariance itself. The invariance related axes embedded in the "invariant information" block in Figure 2 are *measurements*, *patterns* and *scope*.

Effectively making use of and reasoning about a concept requires thoroughly understanding it. Subsection III.B starts by describing a set of common *usage domains* for runtime invariance from the surveyed works. The properties of *invariant information* described in section III.C further describe the different elements of the runtime invariance itself. Finally, the properties of process described in section III.D provide more detailed insights into working with these invariants, related to their *specification* and *evaluation*.

B. Usage Domains

In order to use invariants as a basis to describe and analyse software runtime behaviour it is important to understand the context in which they can be applied. This

includes both the targeted application domain (e.g., test automation or runtime adaptation) as well as the type of application (online or offline). The axis presented in this section aim to provide a basis for understanding what runtime invariance can be used for, how one might use them, and where one might be interested in their application. While this subsection synthesizes the surveyed related works, it should be noted that other application are also possible.

The usage properties refer to the context and type of application of the runtime invariance and are illustrated in Figure 3. This includes both the **usage domains** describing what the invariants are used for, and **usage types** describing if they are applied in an operational system or separately from it. Here the domains are split into five high-level usage domains, which can be observed from two viewpoints: online and offline use. These usage related properties can be observed to help answer questions such as "*How is runtime invariance applied and where?*".

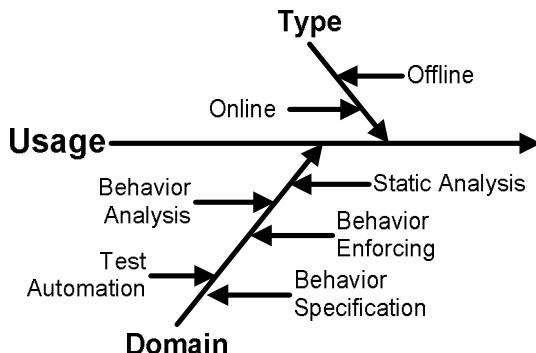


Figure 3. Usage Domains for Runtime Invariance.

Considering the **type** of use, in *offline* use, the invariants are applied separately from the execution of the analysed program. In this case, the observations (collected at runtime) about the runtime behaviour can be analysed external to the operational system. In *online* use the application of the invariants is linked to the executing program. In this case, the invariants are analysed while the system is operational and possibly the results are fed back to the operational system in some form.

Considering the five **usage domains** *behaviour enforcing* techniques guide the online operation of the observed system. *Static analysis* is focused on automated analysis of given static artifacts and thus mainly operates offline. Besides these two, the other domains can make equal use of both online and offline approaches. In the following, each of these usage domains is described in more detail.

Static analysis is not considered in this paper deeply as the focus is on runtime invariance in terms of dynamic analysis. However, some different usage relations can be identified. The invariants can be used as input for static analysis to check if they hold generally or only in specific cases ([24,25]). When information about the program structure is available, the correctness and accuracy of runtime invariants can also be checked and improved with combination of static analysis techniques such as symbolic execution ([26]). Invariants observed as useful and true in

terms of runtime behaviour and dynamic analysis can be turned into more generic checks for static analysis ([14]). Although some specifications of invariants can be more suitable for dynamic and others for static analysis ([27]), many properties of static analysis can also be applied in the context of dynamic analysis and the other way around ([7,28,29]). Finally, as runtime invariants are defined in terms of some formalism, static analysis techniques can also be applied to check a chosen set of interesting properties in their specification, including their correctness and completeness in general and with regards to the observed runtime behaviour ([30]).

Behaviour specification is a basic concept for any application of runtime invariance, as the expected invariants need to be specified before they can be applied. Use cases for invariance in runtime behaviour specification include defining application programming interface constraints (e.g., `size() always >= 0` [25,31]), defining rules (constraints to be obeyed) for successful integration of a component with others ([32,33]), describing the valid (supported) input-space of a component ([32,34]), and defining error handling rules ([35,36]). A component can generally be anything from a method, a composition of classes, a service, or a complete software system.

Invariance also forms a basis for generic formal specification [7], which can be used to verify the actual behaviour against the specific expected behaviour [37]. This also includes constraints for executing a specific functionality [36]. The different properties related to specification of runtime invariance are described in more detail in section III.D.1). A core concept related to this is the requirement to be able to reason about the different properties of potential runtime invariance. A systematic definition for the properties of runtime invariance is needed in order to have a basis for reasoning about their composition and use. Such a definition is given by the taxonomy of invariant properties in section III.C. A specification can be produced either manually, or with the help of an automated specification mining tool.

Behaviour analysis of software runtime behaviour is a human-oriented process typically supported by automated tools. A set of runtime invariants is provided to the user as a basis for analyzing the system behaviour. This can be used for different types of tasks. In the software engineering domain, for example, failure cause location (debugging) can be supported by analyzing how the invariants change over time in an operational system and reporting any significant changes preceding an observed failure ([5,14,29]). This is possible as the runtime invariance of the program can be observed as changing over time. Similarly, debugging can be supported by comparing the invariants observed over both failing and non-failing program executions ([5,38]). Software evolution tasks can be supported by presenting any changes over given invariants when changes are made to a program to make the impacts of changes more explicit ([3,29,39]), such as changed interaction sequences and input-output transformation [39]. Another example in this domain is suggesting refactoring based on mined invariant specifications (e.g., to remove observed constant parameter)

[40], and by using given invariant specifications as contracts to define checks for properties that need to hold after refactoring [41]. Additionally, the invariants can support tasks such as program comprehension by providing a documentation that describes the software behaviour in terms of its important (invariant) behaviour ([3,4,32]). This can also take the form of asking queries to identify invariant sequences to find related sequences of behaviour ([42,43,44]).

When implementing automated software analysis features based on runtime invariance, different approaches for different domains can be taken. In security assurance observing a set of core invariants over specific variables, such as kernel data structures or session state variables, can be used to identify potential security attacks when the expected invariants are violated ([45,46,47]). In the same line, runtime invariance patterns over system interactions (such as library calls) can also be used to identify the origin of mutated code for purposes such as to protect against code theft [48] or to identify mutating malware [49]. Runtime monitoring in general can be implemented to check that the runtime behaviour conforms to the specified invariant specifications in all executions [50,51]. In case errors are observed, error correcting actions can be considered [51], which is a form of behaviour enforcing.

Behaviour enforcing mechanisms take as input the information from behaviour analysis and additionally take automated action to modify the behaviour based on differences in the actual observed runtime invariance vs the specified expected runtime invariance. Automatic adaptation mechanisms can use invariants to choose a new state for the software based on which specified invariants hold at different points in time [13]. Invariants can also be used to ensure that failure states specified in terms of invariants are avoided by modifying runtime behaviour that is observed to be outside the given set of invariants (the expected behaviour) to fit inside the expected invariants ([29,52,53,54,55]). For example, a specific case can be observed in disallowing saving of data or updating the state (of the user interface) when an invariant does not hold [56].

Test automation is basically a comparison of expected runtime behaviour to actual observed runtime behaviour. Defining a test case requires defining test inputs, expected outputs and their relations. Any automated test case basically requires defining the software runtime behaviour in terms of invariance for the automated test case to be able to produce any holding verdicts over the observed behaviour. Thus test automation is a fruitful application domain for runtime invariance.

A basic application is in defining the test oracle (the component that evaluates the test results) in terms of invariance. Such invariants typically define how a part of the system behavior is expected to work, in terms of properties such as determinism ([57,58]), data processing ([4,14,59]), message transitions ([6,60]), and their combinations ([22,61]). This is also related to the previously mentioned aspect of runtime monitoring for correctness in terms of using specified runtime invariance as a set of constantly running online test oracles ([29,35,50,62]). It is also related

to the previously mentioned aspect of specifying how a component should work in relation to its use environment, in using runtime invariance specifications as a means to check how a new or updated component works in different environments ([31,39]).

In addition to test oracles, a test case requires producing valid input for the system under test. Here invariant models can be used to define valid data ranges, value relations and similar properties as a basis for a data model to be used to generate test data to cover these models ([61,63,64]). Another option is to generate test data to try to break previously defined invariants in order to further explore behaviour and to try to extend the model of invariance ([10,59,65,66]). Various approaches to generate test data from these models include search-based algorithms [64], random test generation [63], and generating test data to fulfill the different invariants defined [66].

The above mentioned uses of creating test coverage to cover or break invariants are in themselves also examples of using runtime invariance to reason about test coverage. A second aspect related to this is mutation analysis, which is used to change the SUT and to see how well a set of tests finds the mutations (with failing tests). Invariants can be used to also optimize SUT mutant coverage [10,67], but this can also be applied the other way around to evaluate the quality of the invariants themselves ([49,51]). In this case, the invariants are mutated and executions over these invariants are used to evaluate if the invariants are valid or if they catch useful changes in software behavior.

C. Properties of Invariance

This subsection describes the different aspects and properties related to describing runtime invariance itself. It starts with defining the properties of measurements for the actual runtime observations. Following this, it presents a set of different patterns of runtime invariance that are built from these observations. Finally, the different scopes of runtime invariance that allow for defining where these patterns of runtime invariance can be expected to hold are described.

1) Measurement Properties

As described in section III.A, any evaluation of runtime invariance requires first capturing the required information (observations) about the runtime behaviour. Similarly, as also noted in section III.A, this information can also be used as input in the specification phase. Thus, it can be said that any application of runtime invariance analysis requires also capturing a set of suitable information to describe this invariance. In this paper this information is referred to as measurements.

The basis for describing software behaviour in terms of its runtime invariance is the measurements used to observe this invariance. This in turn requires one to understand what kind of measurements can be made directly from the system, what kind of further measurements can be derived from these basic measurements, and how we may classify all these measurements. This information provides means to describe the system using the higher level invariant patterns presented in section III.C.2). It provides means to create more extensive patterns, and to evaluate the options available and

needed to capture the required information as well as to understand what is needed to instrument the system to acquire these measurements. The axis presented in this section aims to support these goals by providing an overview of what types of measurements are used in existing works.

The properties of these measurements are illustrated in Figure 4. These properties can be observed as helping to answer questions such as “*What kind of basic measurements are used to observe runtime invariance?*”, “*What other measures can we derive from these basic measures?*”, and “*How can we characterize the different measurements?*”.

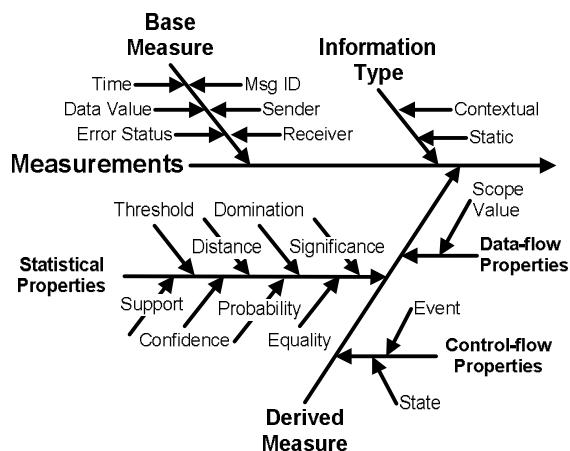


Figure 4. Measurement Properties.

The **information type** of the measurements can be classified into two different types of static and contextual information [28]. *Static information* in a dynamic runtime setting is information that is always the same for a given point of observation. For example, during a specific point of execution, a message passed can always be the same type of a message (e.g., method call named `publishData()`) and is thus static over different executions of this point. *Contextual information* describes dynamic information that changes over the executions of a single point depending on the context (e.g., test case or user session) of the observed information. For example, the time of observation, parameter values, and the thread of execution for a given message all can change over different executions of the same program point ([28,68]). The set of observations can also be grouped ("sliced") according to their contextual information, such as process (thread) id to produce a set of invariants over the scope represented by that slice ([15,28,68]). In this case, the scope (context) identifier becomes the basic measure (e.g., thread id [69] or a constant parameter value [28]).

The term **base measure** here refers to a type of measurement information that describes some basic value of runtime behaviour as it is observed. A basic value for any observation is the *time* when it occurred or was observed ([28,60,63,68]). For dataflow the base measures include variable *data values* and their basic data types such as Boolean values, integers, and character sequences (Strings) [3]. In the scope of object oriented programs the runtime type of an object can also be used as a base measure data value ([14,70]).

From the control-flow perspective, base measures are messages passed between different elements of the control-flow. Perhaps the most basic measure related to this is the *identifier of the message* passed, but also the *sender* and *receiver* objects ([71,72]). Examples of measurement targets include method invocations between components (such as classes or services) ([8,39,71,72]) or invocations on graphical user interface (GUI) operators ([6,61]).

A specific case of control-flow is error handling flows identified by an *error status*. Error scenarios can be classified to generic errors and application specific errors ([6,59]). Generic errors can be related to properties shared by different applications such as database access errors and user-interface (e.g., HTML or DOM tree for a web-application [6]) error codes. When represented in a uniform way (e.g., by programming language exception mechanisms [59]), these can be generally observed in the system behaviour (e.g., by an automated tool supporting a given domain). For example, all Java exceptions can be taken to describe a message that denotes erroneous behaviour being observed [59]. Application specific errors need to be described separately for each application in terms of application specific invariants. For example, one may expect a given error response to a message outside a given set of valid input [22].

A **derived measure** is something that is not directly observed in the system behaviour, but the value of which is rather derived from one or more base measures. To produce derived measures for *data-flow*, the base measures for a system can be grouped based on invariant scopes [3]. For example, the values of variable *x* before and after a program point can be considered separately as variables *x₁* and *x₂*, to describe a pattern saying *x₁>x₂*. In this example, there are two derived measures *x₁* and *x₂*, both of which are scoped data values. The different scopes are discussed in section III.C.3.

Runtime **control-flows** are typically described in terms events and states ([46,61,68,73,74]). These are viewed here as derived measures for control-flow. From this viewpoint, an event can be described as an identifiable, instantaneous action in the observed software behaviour, such as passing a specific message or committing a transaction [69]. Similarly, a state can be described as values of properties that hold over time, such as over interactions between components. This information can be, for example, held in message parameters or inside components internal state variables [46]. A related property is branching, which defines how several different paths of events and states can be taken in the software behaviour. This can be described in terms of invariance of state when observing which paths are taken and which ones are not ([5,75]). In this case, the taking of a branch constitutes an invariant.

Statistical properties describe additional information for other base- or derived-measures. Support and confidence are two values commonly used together ([3,15,28]). *Support* defines the number of times a measure is observed in behaviour ([15,36]). *Confidence* can be used with the same definition [3] but also as a definition for how often another measure is observed in relation to support, meaning how

often a precondition is followed by a post-condition ([15,28,76]).

Probability defines the threshold for a measure to be observed in a given scope, which can be used in different ways. A measure with low probability (support percentage) can be excluded from analysis to address anomalies ([3,13,15,69]). Different approaches are used for this depending on the target invariants, from low level *thresholds* (e.g., 1% or less [3]) to higher levels (e.g., 20% [15]). The probability can also refer to probabilities of a measurement value inside a range of allowed values [13,77]. Deviations from the expected values are typically given a probability which can then define the significance of the deviation ([13,14,46]). This threshold can be used for different purposes such as identifying probable failure causes [14], security attacks [46], possible state transitions [77], and to decide new states for automated adaptation [13].

Equality of objects can be defined in different ways. Besides expecting measures to be exactly equal, semantic equality can also be considered (e.g., two lists can be considered semantically equal while focusing on contained objects and ignoring their ordering [57]).

Distance defines the window inside which two events are observed and considered for evaluating a pattern of invariance. Thus it can be seen as related to the scope of invariance as described in section III.C.3). Events and correlations observed inside a shorter time-window are seen to represent more likely “true” invariance [58]. *Dominance* is a measure used to remove overlapping patterns where one includes the other as a sub-pattern ([78,79]).

Significance defines the importance of an invariant violation or of the measured variable. With regards to invariant violations, different approaches to significance can be taken where the latter observed violations are given higher priority as they are seen to be closer to a failure [14], or earlier violations as they are expected to have more impact on later behaviour [53]. When a variable is observed as having no correlation with other variables it can be considered as irrelevant for analysis purposes ([31,40]). The significance of observations and invariants can also be defined according to the number of observations ([13,58]).

2) Patterns

Having a set of measurements available in itself is not useful alone. They provide a basis for analysing the runtime invariance of the system but do not tell much about the invariance itself. A statistical derived measure can sometimes be useful in terms of considering basic runtime invariance in terms of single measurements at a single point of execution. However, more complex patterns describing relations between the measurements in different scopes (e.g., over time, described in more detail in section III.C.3)) are also needed. This includes considering their relation to the overall control-flow in terms of their occurrence in the given scope, their concurrent interleaving and sequential ordering. It also includes considering how the values interact in terms of data-flow and whether a specified invariance should be expected as normal or exceptional behavior of the system. Knowing these more advanced patterns enables describing and analysing the runtime invariance more effectively. The

axis presented in this section aims to support these goals by providing an overview of what types of patterns are used in existing works.

Analysis of runtime invariance is basically about analysing patterns of invariance over the observed behaviour. The set of such patterns identified in this paper is shown in Figure 5. In relation to the different types of measurements described in section III.C.1), control-flow related patterns describe ordering of events or states in the observed system ([7,63]). Data-flow related patterns describe the data-flow of the observed software, such as what values a given variable takes during the software execution ([3,74]). These patterns can be observed as helping to answer questions such as “*How are the different measurements grouped?*”, “*What are their relations to each other?*”, and “*What type of behaviour do they describe?*”.

Together these can be combined to represent the overall behaviour of the software in terms of the control-flow combined with the data-flow. A basic way to describe these combinations is in terms of conditional dependence; a control-flow event can only be followed by one of many (branches) depending on a given condition ([8,27,73,76,78,79]). A natural way to express these conditions is then in terms of invariants related to the data-flow in the context of that control-flow. For example, event P1 can be followed by event P2 when $x < 0$ or by P3 when $x \geq 0$ ([8,22,78]).

Overall, these are referred to here as behavioral invariants, where the constraints for a given control-flow pattern are defined in terms of its data-flow invariants. These can be described in terms of models at different abstraction levels as discussed in section III.D.1).

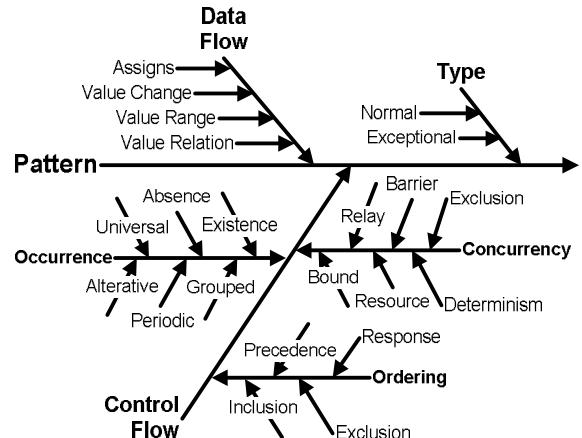


Figure 5. Patterns of Runtime Invariance.

Each pattern can further be related to describing different **type of behaviour**, which can be generally classified as *exceptional* (error) or *normal* (correct) behaviour of the observed system ([6,25]). Errors can be classified as either persistent, in being possible to identify them at different points, or as transient, in which case they are not observable after some set of events [80]. Exceptional states can also refer to more than just error situations, such as behavior deviation and need for adaptation [13]. A basic approach to

detection of exceptional states is to look for deviations of a model describing the “correct” behavior, and to act on the observed exceptions with a predefined strategy ([6,13,61]).

Patterns related to runtime invariance of **control flow** can be defined as describing the sequential dependencies between a programs events and states ([7,76]). In the following discussion the term “event” is used to refer to both events and states. Here, the control-flow patterns are classified to three main categories related to **concurrency**, **occurrence** and **ordering**.

Concurrency in runtime analysis can be categorized as an event stream produced from several sources at a time [76]. *Determinism* defines that a set of operations always produces the same result, even if performed with different parallel schedules, as long as the start state is the same [57]. *Bounded* refers to allowing at most N threads to execute at one time in a single block [81]. This can also be referred to in terms of overlapping or sequential blocks [69]. *Exclusion* is a pattern only allowing one thread (from a set of threads) in a single block (or a set of blocks) [81], and can also be referred to as mutual exclusion ([61,69]). *Resource* is a pattern where a thread can execute only if enough resources from a resource pool shared by a group of threads are available [81]. A *barrier* pattern relates to two or more threads having to wait at the edge of a synchronization block to exit at the same time [81]. *Relay* is similar to barrier but allows exit of thread t1 from region r1 after thread t2 enters r1 [81]. Both relay and barrier can also be generalized to groups of threads (or event sources) arriving and leaving [81]. When these constraints of invariance are not met, problems can occur. For example, if two or more events affect the same state asynchronously, they typically will corrupt the overall application state if executed concurrently [82].

Occurrence related patterns describe properties related to observing an event or a state [7]. The *grouped* pattern describes two or more events always appearing together regardless of their ordering ([58,76,83]). For example, the methods `setHost()` and `setPort()` can be called to set up properties for a connection in any order but must always appear together [58]. The grouping can be seen in relation to context, coupling an event to its context in allowing only certain events in a certain context (state) [83]. *Absence* defines an expectation that the measure does not exist in the defined scope [7]. *Existence* denotes that the measure exists in a scope [7], and can be extended as bounded existence defining that the measure exists N time in a scope, where N denotes either exact, minimum or maximum number ([7,78]). *Universal* defines an expectation that the measure applies to the whole scope ([7,29,84]). *Periodicity* describes a measure repeating over a given cycle (scope) ([76,85]). The *alternative* pattern defines a set of events out of which one should be observed in a given scope ([86,87]).

Ordering describes the patterns of order between the different events ([7,88]). Precedence describes a specific event P always occurring before another specific event Q [7], also referred to as a precondition ([27,29]). Any number of events can also be observed between the two events ([15,58,63]), and it is possible to define the number of

allowed events in between [58,84]. A specific case of this is chain precedence, which specifies that a sequence of events (Q₁, Q₂, Q₃, ...) is always preceded by another sequence of events (P₁, P₂, P₃, ...) [7]. This can also be related to an event enabling or disabling another on in the future [89].

The opposite of precedence is *response* which defines that event P is always followed by event Q ([27,29]). Similar to precedence, also here any number of events can be observed between the two events ([15,58,63]), and it is possible to define the number of allowed events in between ([58,84]). The scope for response can be defined in different terms such as inside a given time duration [85]. This is again a specific case of chain response, which defines that a sequence of events (P₁, P₂, P₃, ...) is always followed by another sequence of events (Q₁, Q₂, Q₃, ...) [7]. This can also be related to relation of objects to events, such as a created object always being passed as a parameter or an object that is related to event A also being related to event B [90].

Related to the precedence and response patterns, and also to the grouped occurrence pattern, two or more events can also be grouped together as an alternating sequence such as ABABAB ([76,78]). Further, it is also possible to define other more specialized cases such as events in the alternating sequence repeating multiple times, AB^{*}C, where B is repeated 1-N times between A and C [78], or a cutoff in the end of the sequence (ABABA) [15].

Inclusion can be used to define an event always appearing inside another, where the dominating event must then be defined in terms of a larger scope ([63,85,88]). For example, an event A can hold while a specific state B holds [88], or when one is observed, another must also hold for a given scope [85]. *Exclusion* is the opposite of inclusion and defines that when a dominating event holds, a specific other event cannot hold. For example, when state B holds, event A is now allowed ([61,83]). Related to this and also the alternation pattern, it is also possible to define that when event A holds, event B does not, but when A no longer holds, B must hold [88]. Some basic examples are disallowing communication with a thread that is not started [91], or a model dialog disallowing communication with other dialogs (states) [61].

Patterns related to runtime invariance of **data flow** describe properties and relations over variable values during program execution ([3,74]). The *assigns* pattern defines that in a defined scope, values of specific variables are assigned to (modified) ([3,25]). This can also be described in terms of values that are not modified [3]. *Value change* is an evolutionary pattern that describes how a value changes over time in a given context ([14,27,92]). This pattern defines the expected scale of change for a variable in a given scope. For example, the expectation can be that change in value is always small (within a given threshold such as `change<5`) ([14,92]). Examples of specific case are a variable that is never set (`null` value) ([3,27]), and a value that is generally constant in the given scope [27].

A *value range* describes a variable always having a value inside a defined range in a given scope (e.g.,

[3,14,27,33,56]). Examples include value always being constant, one of a set of possible values (e.g., one of 1, 2, 4) and a value between given boundaries (e.g., $1 < x < 4$) ([3,14,86]). Common constants such as zero or one can also be considered a specific case in itself ([3,14]). Optimizing for performance a subset can also be selected such as looking for positive ($x > 0$) or negative values ($x < 0$) [14]. Another example is that the contents of a character string are expected to be a human readable character ([27,64,93]). This can be further extended with a probability distribution describing how often each character is expected to be observed [46]. For user interfaces, it is also possible to define a possible set of UI widgets and their values ([56,61]).

A *value relation* pattern describes how one variable is related to another ([3,31]). These can be basic mathematical operations (e.g., $x < y$ or $x = y + 1$), or their combinations [3]. Relations can also be described in terms of the relation of one variable to several others, such as the relation of program output to its inputs [31], or as a variable always belonging to a larger set (member of another array variable) [3,27]. In the case of larger sets of values (e.g., arrays), the same relations can be described internally between the elements of the set [3]. Additionally, a set of specific relations can be considered such as one set (array) reversing another or matching a subset of a bigger set [3]. Additionally, a single value (e.g., a given variable or a constant) can be described to always be included in a given set [3]. The evolution of different variables can also be linked so that when the value of one changes, the other must change also [94].

3) Scope of Invariance

While the patterns described in section III.C.2) describe the basic important properties of runtime invariance, simply defining these patterns without defining when they are expected or observed to hold is not sufficient. To effectively describe the runtime invariance of a system, it is also important to be able to define when this invariance is expected to hold. This scope can be defined in terms of events or time specifying the constraints for when the pattern should hold. The axis presented in this section aims to support these goals by providing an overview of what types of scopes for different patterns are used in existing works.

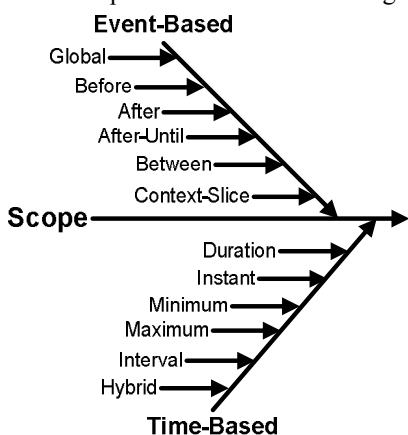


Figure 6. Scope of Invariance.

The scope of an invariant defines when and where a specific pattern of runtime invariance is expected to hold. This scope element of the taxonomy is shown in Figure 6. This part of the taxonomy can be observed to help answer questions such as “*Where and when do the patterns of runtime invariance apply?*” and “*What defines these scopes of runtime invariance?*”

In the following descriptions, the term event is used to refer to both control-flow events and states and data-flow measures. The scope is split into two main categories of time-based scope and event-based scope, which are discussed next.

An **event-based** scope defines the scope in terms of relations between events and observations. An invariant may define that it should hold *after* a given event [7]. Specific cases of this are defining the tail of an event set, where on N last events are considered [3], or when an event is never expected to occur after a given observation [88]. For example, using the tail scope, the relations between the last 2 observations can define how a value in a set increments [3]. More specifically than after an event generally, another event may be defined as the end condition in which case the invariant should hold after the observed start event until the observed end event (termed as the *after-until* scope) [7]. This can also be defined as the state holding true until a specific event is observed or forever if the end condition is never met [84]. A similar scope is *between*, which defines two events in between which the invariant pattern should hold [7]. However, the difference is that this holds only once both the start and end events have been observed, and after-until holds from the first observation of the start event [7]. Specific examples of these are the start and end of a method invocation on a component ([3,46]).

As opposed to the after scope, an invariant pattern can also be defined to hold only *before* a given event is observed [7]. Similar to the tail for the after scope, here the first N observations of a set can be considered with the term of head [3]. A *global* invariant pattern should hold for all observed behaviour during the program execution [7]. The scope can also be defined in combination with a specific slice of the program behaviour, such as a thread ([28,68]) or a specific web application session [46]. In this case the scope becomes a combination of the *context slice* and one of the other scope definitions discussed above.

A **time-based** scope defines the scope in terms of relation of the observations to the passage of time. Different aspects of the *duration* of time can be used to define the scope in terms of time. The basic form can be defining the start time and length [63], or defining an event as *instant* without specific timing constraints [69]. The duration can also be specified as an *interval* with a *minimum* or *maximum* time [88]. It is also possible to define a *minimum* or a *maximum* time (duration) for a pattern to hold without specifying the other bound [85]. Additionally, the time duration can be combined with an event-based scope to produce a *hybrid* scope. For example, an invariant pattern can be defined to hold after 10 time units (according to choice of time unit) until a specific event is observed [88].

D. Process Properties

In addition to the properties of invariance itself, it is useful to consider the properties of the process used to work with these invariants. The properties of the process in this section are described in terms of two main axes as noted in section III.A. These are the properties related to specification and evaluation of runtime invariance. The third step of measurement as described in section III.A is most closely related to the invariant information described in section III.C and thus not covered in this section.

1) Specification Properties

In addition to understanding how and where the runtime invariants are applied, it is also important to understand how runtime invariance can be expressed and where the information to describe a system in terms of this runtime invariance comes from. This helps to choose effective means both to specify the expected runtime invariance for a system, and to express it in suitable and effective terms for analysis. The axis presented in this section aims to support these goals. This provides a basis for specifying invariants using the detailed properties of runtime invariance presented in section III.C.

The different aspects related to the specification of runtime invariance are illustrated in Figure 7. The **expression** axis describes the different aspects relevant to how the invariants are expressed in specifications, including the expression *language* and the *abstraction level* of the expression. The specification *source* describes the different sources of information for the specification. The *method* of specification defines how the specification is created. These different properties can be observed to help answer questions such as “*Where does the information to define runtime invariance come from?*”, “*How is the runtime invariance defined and reasoned about?*”, and “*How is runtime invariance expressed?*”.

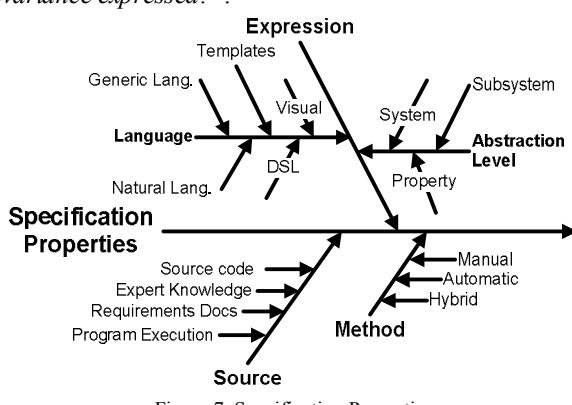


Figure 7. Specification Properties.

Different types of **methods** can be used to obtain the information for specifying the invariants. One is to fully (*automatically*) reverse engineer these from observing program behaviour ([3,38,80]), commonly referred to as specification mining. The opposite of this is describing the invariants *manually* based on specifications or expert knowledge ([6,29,80]). Finally, a *hybrid* approach can be taken, where an initial specification is first generated with

tools similar to the automated approach, and this reverse-engineered information is manually augmented with information from specifications and as feedback from continuously evaluating this preliminary model ([22,37,42,71]).

As noted above, the method of specification is closely related to the **source** of the information used for the specification. Natural language documents such as *requirements specifications* can be manually analysed to find a set of relevant invariants [22]. *Expert knowledge* about the system behaviour can also be used to specify runtime invariance at different abstraction levels, for example, specified by domain experts as describing high-level system behaviour [56], or as lower level invariant properties specified by developers with detailed knowledge about the implementation [29].

Source code and program execution are two sources of information most suited for automated analysis. *Source code* is a static artefact, but where available can also be used as an additional input for dynamic analysis such as providing interface definitions ([22,44,95]), or to provide additional information for assisting in dynamic analysis [26]. *Program execution* is observed in terms of dynamic analysis to capture how the observed system behaves in a given context such as a test case [2].

From the different sources of information, one needs to capture a set of invariants covering the relevant properties of dynamic behaviour in the software. Experiments have shown that combining both manual and automated sources of information gives the best results, where both provide useful invariants not identified by the other approach [96].

Expression needs to define the invariance using a suitable expression language, and at the chosen abstraction level. The *abstraction level* of the specification can be described as defining the overall execution of the system or focusing on specific parts of execution ([88,97]). This definition at different levels is also described in terms of different types of *languages*, such as automata for overall behaviour and assertion style specifications for specific properties ([37,98]). A relevant concept for definition of runtime invariance for a component is the hierarchical relations of the different components, where a subtype can be viewed to also inherit the invariant definitions of its supertype ([29,34]).

In terms of the *abstraction level*, different focuses for partitioning the modeling can be taken. For example, models can be classified at the implementation, design, and domain level [98]. This can be translated to the internal implementation of components (e.g., embedded checks for specific *properties* [27,29]), the external interfaces of components as *subsystems* ([35,50]), or the overall *system* (domain) behaviour ([8,61]). Different types of approaches can be taken that combine different viewpoints from these. For example, some approaches define small-scale state-machines for specific parts of execution ([60,62,99]), which are then combined to form larger wholes to be checked, with the expectation that the larger whole needs to be covered ([18,60,78,99]).

A benefit of a small-scale specification can be seen in making it easier to produce a suitable formal specification for a specific property ([51,57]). Small (more specific) invariant definitions are seen as more suitable for specific low-level checks, but for system-level checks they can become too complicated and interleaved ([51,99]). In this sense, a different type of an approach such as specifying of larger-scale state-machines is seen as more appropriate ([51,99]). To reduce the complexity of managing specific checks some recommend defining fewer checks, and focusing on effective specifications that embed the important elements of behaviour ([27,51]). In this way, the larger specification can also be composed of several smaller ones [99].

For a higher-level specification a common model is different forms of a state-machine ([8,61,89,93]). Guards can be defined in terms of invariance over data values ([8,33]). Similarly, the possible transitions can be viewed as invariants in possible actions in a specific state ([8,61,89,93]). Many approaches in analyzing the state-machines are similar to how low-level invariants are combined to form state-machines. In this case, high-level state-machines are decomposed into smaller invariants for various analysis purposes, such as using expected transitions of chosen length as test oracles ([89,93]).

To support the different types of specification methods, the *expression language* needs to be both formal to allow for automated tools to effectively process them but also understandable to a human user in order to also support their manual review and analysis where needed. *Domain specific languages (DSL)* can be used to describe the invariants specifically for a chosen domain, such as in form of test oracles for web-applications ([6,56,98]). Other application domains include analysis support for refactoring [41], synchronization [81], determinism [57], general temporal properties [88], and test definitions in terms of runtime invariance [100]. Besides direct support, specific domains from more generic models can also be supported through model transformation (e.g., to monitoring code for runtime correctness [83]). Many of these languages are specifically designed to support modeling of the chosen set of properties for runtime invariance ([41,56,57,60,63,81,88,100]).

Besides these specific expression languages, also *generic languages* can be used. For example, the object constraint language from the unified modeling language can be used to express invariants ([101,102]), or live sequence charts to express invariance in event ordering [43]. Logical expressions can be used as a basis for defining invariance in themselves ([42,84,85]), or as part of the specific languages [41]. Regular expressions are generic expressions intended to express patterns over strings, and thus form a natural basis for expressing also patterns of runtime invariance. In this case, the event stream is expressed as a string of events, and regular expressions are used to express patterns over these events ([58,100]). Perhaps the most expressive means to define invariant properties is in terms of programming languages, which allows using their full expressiveness to define the invariance. This can take different forms, such as interleaving with the implementation code to be compiled

with the program itself ([25,29]), writing them separately as a basis for a separate monitoring program [35], or as “model programs” for how parts are expected to behave allowing to execute the invariant definition separately with or with the implementation to perform different analysis and checks [50].

A common approach to support the definition of invariants is in using *pattern templates*. This is especially true in the specification mining approach, where a set of templates are defined and reflected against the actual observed behaviour to report observed invariants matching the template definitions ([3,14,58]). Templates are also used in manual specification ([41,103]). In both cases, the templates describe a predefined set of patterns, which are then parameterized according to the expected or observed runtime behaviour. The set of patterns of runtime invariance is described in the section III.C.2) of this paper.

As mentioned previously, an important aspect of specifying runtime invariance is that the specification should support both manual analysis and processing by automated tools in relation to the methods of working with these invariants. The languages described above are mainly focused on effective description from the automated processing perspective. One approach to address this is to produce specifications in a programming language when targeting developers in order to provide a familiar language to reason in [37]. Along with the different language transformations discussed before (in relation to [83]), it is also possible to provide transformations into *natural language* to support easier comprehension of the specifications (e.g., into structured English [85]). A related specification approach is also that of grammar-based specification, which aims to describe the invariance in terms of sentences of events [99].

Formal textual specification of complex behaviour has also been shown to be very hard for humans [103]. To address this, besides using textual languages and transformation between them, *visual* representations can also be seen as a more natural way of expressing invariance for humans ([56,103]). In this case, the transformation is done from the visual representation to a more machine processable form.

2) Evaluation Properties

In relation to this overall process flow described in section III.A, the overall process of runtime invariance evaluation is also related to the step of measurement and typically consists of three distinct steps: Collecting information (a set of events) from the runtime execution of the program, building a model of runtime invariance based on these observations, and comparing this model against the specified reference model of runtime invariance ([88,93,104]).

Specific aspects to consider in evaluating the runtime invariance include how extensive evaluation is done in terms of depth and frequency, the cost-effectiveness of these choices, and how the evaluation check is performed. Effective evaluation requires considering these different aspects together when building the evaluation for the case at hand. The axis presented in this section aims to support these

goals by providing an overview of the different properties related to evaluation of runtime invariance in existing works.

The evaluation properties are illustrated in Figure 8. This is described in terms of three main properties. The **depth** of evaluation defines how extensively evaluation is performed. **Triggers** are events or states that trigger the evaluation of a specific runtime invariant pattern. Finally an evaluation function needs to be defined to **check** each defined invariant property. These properties can be observed to help answer questions such as “*How is runtime invariance evaluated?*”, “*How extensively is it evaluated?*”, and “*What is the impact of different properties of evaluation?*”.

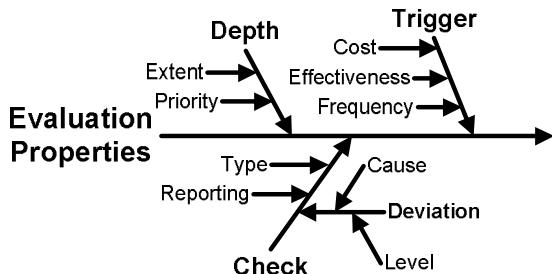


Figure 8. Evaluation Properties.

The **depth of evaluation** defines how extensively evaluation is performed. One approach is to define a *priority* to each defined check and to choose when an evaluation is performed to evaluate only checks of a given priority or higher [27]. Different properties for different types of components can be evaluated at different levels, such as checking a library while in development thoroughly and after release, focusing in more detail how the clients interact with it [29]. The evaluation level can also be defined in terms of evaluated data, such as if input is given, if it is of correct type and if it meets the domain-specific checks defined for it [56]. The breadth of evaluation can also be defined in terms of which components are checked, such as checking only properties of the active component, all components in the scope, or all components accessible [105].

The **check** defines the evaluation function for each specified property of runtime invariance. Similar to the level of evaluation, it is also possible to define the severity *level of deviations* from expected invariance, such as informative, warning and error [56]. When a deviation is observed, the *cause of deviation* can be classified to one of three options: the original specification was lacking due to insufficient input, the observing a feature in a specific context or environment that was not previously considered, or that there is a real failure observed in the runtime behavior [67]. The checks can be classified to two main *types*: the property must never be violated, or the property must always hold [84]. Different approaches to *reporting* the status of violations can be taken, such as reporting only violations [58], reporting also passing checks with chosen filtering criteria [28], or reporting the status of all checks. The threshold for when a deviation from the specification is considered significant enough to be reported can also vary according to the properties of specified runtime invariance (as discussed in more detail in section III.C), such as reporting only

violations for a property when a certain number (threshold) of violations for that property have been observed [58].

The evaluation **trigger** defines when the evaluation is performed. The *cost effectiveness* of evaluating invariants is related to the frequency and depth of their evaluation ([60,97]). Different options for *frequency* can be, for example, to check a single property for a single component or for all components after each event, to check all properties after each event, or to check all properties after larger execution points (e.g., test case [97]). Some trade-offs to consider include checking a single property not always revealing errors [97], performing checks often helping in finding “transient” errors that are no longer visible in later states [80], and the increased cost and power of fully performing all possible checks at all times [97]. One approach is to focus the checks on a specific set of chosen events and properties [56].

IV. EXAMPLES

This section illustrated mapping the taxonomy presented in the previous section to practical concepts in terms of two different types of runtime behaviour modelling approaches. The first one describes modelling specific aspects of runtime invariance in a distributed sensor data collection system. The behaviour of interest in this case is event-focused, where patterns over sequences of interactions are important and should be observed to hold and are observed while the system is operational (online). A second example is provided in terms of a data visualization application. In this case, the data-flow aspects are more important and the invariance is analysed separately from its operation (off-line).

The examples presented here are intended to illustrate how different properties of runtime invariance in software behaviour can be defined for different systems. From this, different approaches can be taken to build required support for different usage domains. The details for this support are left for specific works in those domains, while we illustrate a set of specific invariant properties for each.

A. Case 1: Sensor Data Collection

This case example describes a mapping of the taxonomy for describing the runtime behaviour of a system in terms of a model-based testing tool called OSMOTester [106]. The target system is a sensor-platform server-node that manages a set of sensor-nodes. This section uses as an example a single feature related to keeping track of the dynamic non-persistent runtime state of the node.

As one of the main properties of upholding this state, the server needs to keep track of all available sensor nodes that register to the system. The sensor's registration is by sending a specific message. After this, the constant keep-alive messages need to be provided to uphold the registration. If one is not received for duration of 10 seconds from a registered sensor, it is removed from the list of connected sensors and a matching event is generated to inform any interested clients of the sensor platform.

Considering the process perspective of the taxonomy, this is mainly in the test automation usage domain. The tool we use applies a generic programming language (Java) to

model the behaviour so this sets the expression language. For abstraction level, our focus with this example is on a specific behavioural property, which could be extended in following iterations to include further properties separately or integrated into a subsystem model. The specification approach in this case is manual specification with the help of expert knowledge and documentation, although tools such as Daikon could be used to provide invariants over observed executions as input as well. From the evaluation perspective, in the case of test automation, we wish to evaluate all relevant aspects for the property under analysis, report only the deviations and their cause. These aspects define the process related properties for this example in relation to the taxonomy.

Considering the properties of invariance perspective, we can define as basic measurements the registration messages, keep-alive messages, the sender of the messages (the sensor node) and the time of receiving any message. Of these, the messages themselves are static information, whereas the timestamps and the sender are contextual information. As an example pattern of invariance, we can define an expectation as “Keep-alive observed continuously after registration with a minimum of 10s interval”. Several other patterns could also be formulated, including the timeout and first registration itself. These aspects define the invariant information related properties for this example in relation to the taxonomy.

B. Case 2: Data Visualization Tool

This second case example describes a mapping of the taxonomy for a tool used to help visualize and analyse behavioural data collected from the execution of a system. This section uses as an example a single feature of observing historical data over time. This feature allows taking several different measurement values and comparing their evolution over time independently and in relation to each other using graphical visualizations.

Considering the usage domain and process perspectives, in this case, the analysed data is not captured and used in real-time but rather the tool is used to load data from a log file, making this practically an offline approach. The application domain is specifically behaviour analysis, and the applied representation language in this case is a visual language in terms of graphical representations. The abstraction level depends on which part of the system is described in the log file, but is typically the overall system behaviour. The trigger and depth of evaluation are in this case related to the information captured during the runtime logging step, and depends on how the system is instrumented. The evaluation checks are performed by the human user based on the visualizations and their manipulations.

Considering the properties of invariance perspective, the basic measurements include time and data values. The relevant patterns are mainly data-flow oriented due to analysing data value evolution over time. This includes value change as observed over time for a single variable, and value relations as observed across the different visualized values. As the visualization produces a separate graph for each value, each of these is a contextual slice and the scope of

these graphs is always time-based. We can further define derived measures as statistical properties of the observed data and use those to define events that can be used to define further scopes for patterns and comparison.

V. DISCUSSION

The taxonomy and its classes presented above are based on the existing work in the literature. In this sense it limits itself to discuss properties only relevant to those in the chosen works. Additionally, it is possible to use and explore other possible relations. For example, many of the described control-flow patterns also apply to data flow patterns. For example, a value may be defined to precede another value (relating to the precedence control-flow pattern). Similarly, the set of data-flow patterns can be considered to apply in the context of control-flow. For example, the range of possible control-flow options following one control-flow event can be in a given range of possible defined control-flow events or states (related to value-range data-flow pattern).

It is also possible to take different viewpoints on the different properties discussed categorized in the taxonomy presented in this paper. For example, the taxonomy lists a set of time-based scopes, and a set of patterns related to the ordering of events. In other cases, for example, Konrad and Cheng define a set of patterns such as bounded response where a reply is expected in a given timeframe [85]. This is an example where a set of properties presented in the taxonomy in this paper are combined to form a specific set of patterns in a given domain. Specifically, the taxonomy presented in this paper aims to decompose these into their constituting parts that can be composed in different ways. While this is more generic and provides a basis for wider application, in practical application in different domains, it may be more suitable to specify a set of more specific patterns such as those defined in [85]. In this case the taxonomy can be used as an aid to create the set of suitable patterns.

Overall, the discussion in this paper is from the generic viewpoint of using runtime invariance. When a set of invariants are defined for a system, one important aspect to consider is how representative these are in describing the relevant properties of software behaviour. When defined manually by an expert, the invariants can be expected to describe relevant and important properties. However, even in these cases important invariants can be missing and in many cases no invariants are defined at all. In these cases, automated inference techniques can be used to assist in finding invariants. Both of these cases have been shown to be valid as also discussed in section III.D.1). Improving the means to help manually define invariants and to automatically mine for relevant ones thus is an interesting research question. Potential approaches to investigate include using a set of chosen invariants known to be interesting in the given domain, using combined information from static analysis, relying on statistical values to report the more interesting ones, and providing more advanced support for combining both the manual and automated approaches as also discussed in section III.D.1).

Discussion on the statistical properties of different patterns and measures highlight differences in the applied approaches. For example, in many cases the invariant patterns that have only low support level (i.e., there are few cases) are not reported. In the extraction phase, this can be useful in removing patterns observed merely due to chance that may be incorrect in themselves due to interleaving of concurrent behaviour, or completely irrelevant in the general context [3]. On the other hand, sometimes all observed behaviour is important regardless of their probability. This can be, for example, behaviour that is only rarely observed in the observed executions but is still equally important for the overall system behaviour (e.g., error handling or corner cases) [22].

Use of invariants in different domains as discussed here is not limited to those aspects discussed. In fact, many systems use invariants for various purposes but these are not always discussed in terms of invariance. For example, as discussed in section III.D, in test automation the test oracle practically always needs to be described in terms of invariance, where the input is expected to produce a given output (the relation of input to output should be invariant). In this sense, defining invariants as discussed here can be beneficial in a wider context of how people think about the behaviour of programs. However, presenting a meaningful language to describe the invariants and use them in different contexts, as well as furthering people's understanding of their relation to different domains, can be required for adopting them as a concept more widely. Use of domain-specific languages and related tools is an option for this.

Understanding and using invariants generally requires specific considerations for specific usage purposes. For example, one may refactor code based on suggestion from invariant analysis [40] but this also needs to consider the part where the human user needs to read the code and understand it. If the refactoring reduces this understanding by hiding information, this refactoring may be more harmful for the overall software maintenance. Similar needs for understanding the invariants in general ought to be considered.

Application of different properties depends on the target domain as well as the process it is used to support. For example, in testing it may be more appropriate to report all results of invariant evaluation. On the other hand, in runtime (online) monitoring of an operational system it is more useful to optimize the evaluation approach to minimize the intrusiveness on the system.

The case examples presented in section IV are from actual applications of the taxonomy, where different people have applied the taxonomy with the help of the author. In these cases, the typical approach has been to start with some specific properties and progressing from those iteratively to include more functionality. Thus, an iterative approach of adoption from the taxonomy can be seen as a useful process for its application. However, more extensive case studies in software behaviour analysis and application of the taxonomy are left as a topic for future work.

These case examples also illustrate how the taxonomy could be used as a basis for building domain-specific

languages to support analysing different topics of runtime invariance. For example, in modelling system behaviour in terms of runtime invariance for test automation as presented in the first case example, a domain-specific language could allow composing modelling components based on the different types of measurements, patterns and scopes into test models. Similarly, in the second case example, the properties could be used to provide more generic base for features to manipulate and visualize the measurement data in terms of derived measures and patterns. These studies are left as a topic for future works.

VI. CONCLUSIONS

Today, runtime invariance is used in the context of many aspects of software design and analysis. The invariants for different systems are as different as their behaviour, but this paper has collected a set of common properties from existing works and presented a taxonomy describing these common properties. This should help give a more common understanding of runtime invariance in software behaviour and help in using invariants to describe it in different domains.

The presented taxonomy is based on six main facets, one describing the usage domains, two related to processes of using the invariants and three related to the information describing the invariants themselves. Overall the focus can be defined as describing the invariant information in the context of the process.

The main contribution of this paper is presenting the underpinning of a classification overview for understanding the space of runtime invariance. This provides a basis for more thorough reasoning about invariants, building tool support and identifying future research questions. Some specific questions identified include possibilities of providing more focused domain-specific invariants on top of the taxonomy and providing more extensive tool support for using the invariants according to the taxonomy presented, as existing tools only consider parts of it.

ACKNOWLEDGMENT

The author wishes to thank Arie van Deursen, Ali Mesbah, and the anonymous reviewers in the PATTERNS 2010 conference for their useful comments and discussions on the pilot study. The author also wishes to thank Lars Ebrecht for the interesting discussion and comments on the extended version of the taxonomy.

REFERENCES

- [1] T. Kanstrén, "Towards a Taxonomy of Dynamic Invariants in Software Behaviour," in *2nd Int'l. Conf. on Pervasive Patterns and Applications (PATTERNS 2010)*, Lisbon, Portugal, 2010, pp. 20-27.
- [2] B. Cornelissen, A. Zaidman, A. van Deursen, L. Moonen, and R. Koschke, "A Systematic Survey of Program Comprehension through Dynamic Analysis," *IEEE Transaction on Software Eng.*, vol. 35, no. 5, pp. 684-702, 2009.
- [3] M. D. Ernst, J. Cockrell, W. G. Griswold, and D. Notkin,

- "Dynamically Discovering Likely Program Invariants to Support Program Evolution," *IEEE Transactions on Software Eng.*, vol. 27, no. 2, pp. 99-123, Feb. 2001.
- [4] M. Boshermitsan, R. Doong, and A. Savoia, "From Daikon to Agitator: Lessons and Challenges in Building a Commercial Tool for Developer Testing," in *Int'l. Symposium on Software Testing and Analysis (ISSTA 2006)*, Portland, Maine, 2006, pp. 169-179.
- [5] T. M. Chilimbi, B. Liblit, K. Mehra, A. V. Nori, and K. Vaswani, "HOLMES: Effective Statistical Debugging via Efficient Path Profiling," in *Int'l. Conf. on Software Eng. (ICSE 2009)*, Vancouver, Canada, 2009, pp. 34-44.
- [6] A. Mesbah and A. van Deursen, "Invariant-Based Automatic Testing of Ajax User Interfaces," in *Int'l. Conf. on Software Eng. (ICSE 2009)*, Vancouver, Canada, 2009, pp. 210-220.
- [7] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett, "Patterns in Property Specifications for Finite-State Verification," in *Int'l. Conf. on Software Eng. (ICSE 1999)*, Los Angeles, CA, USA, 1999, pp. 411-420.
- [8] D. Lorenzoli, L. Mariani, and M. Pezzè, "Automatic Generation of Software Behavioral Models," in *Int'l. Conf. on Software Eng. (ICSE 2008)*, Leipzig, Germany, 2008, pp. 501-510.
- [9] L. A. Clarke and D. S. Rosenblum, "A Historical Perspective on Runtime Assertion Checking in Software Development," *ACM SIGSOFT Software Engineering Notes*, vol. 31, no. 3, pp. 25-37, 2006.
- [10] D. Schuler, V. Dallmeier, and A. Zeller, "Efficient Mutation Testing by Checking Invariant Violations," in *Int'l. Symposium on Software Testing and Analysis (ISSTA 2009)*, Chicago, USA, 2009, pp. 69-80.
- [11] R. Floyd, "Assigning Meaning to Programs," in *Symposium on Applied Mathematics*, 1967, pp. 19-32.
- [12] C.A.R. Hoare, "An Axiomatic Basis for Computer Programming," *Communications of the ACM*, vol. 12, no. 10, pp. 576-580, 1969.
- [13] L. Lin and M. D. Ernst, "Improving the Adaptability of Multi-Mode Systems via Program Steering," in *Int'l. Symposium on Software Testing and Analysis (ISSTA 2004)*, Boston, Massachusetts, USA, 2004, pp. 206-216.
- [14] S. Hangal and M. Lam, "Tracking Down Software Bugs Using Automatic Anomaly Detection," in *Int'l. Conf. on Software Eng. (ICSE 2002)*, Orlando, Florida, USA, 2002, pp. 291-301.
- [15] D. Lo and S. Khoo, "SMArTIC: Towards Building an Accurate, Robust and Scalable Specification Miner," in *Int'l. Symposium on Foundations of Software Eng. (FSE 2006)*, Portland, Oregon, USA, 2006, pp. 265-275.
- [16] B. Kitchenham, "Guidelines for Performing Systematic Literature Reviews in Software Engineering," Keele University, Keele, Staffs, EBSE Technical Report 2007.
- [17] S. MacDonell, M. Shepperd, B. Kitchenham, and E. Mendes, "How Reliable are Systematic Reviews in Empirical Software Engineering?," *IEEE Transaction on Software Eng.*, vol. 36, no. 5, pp. 676-687, Sept./Oct. 2010.
- [18] S. Ducasse and D. Pollet, "Software Architecture Reconstruction: A Process-Oriented Taxonomy," *IEEE Transactions on Software Eng.*, vol. 35, no. 4, pp. 573-591, 2009.
- [19] H. Kagdi, M. L. Collard, and J. I. Maletic, "A Survey and Taxonomy of Approaches for Mining Software Repositories in the Context of Software Evolution," *Journal of Software Maintenance and Evolution*, vol. 19, no. 2, pp. 77-131, 2007.
- [20] M. S. Ali, M. A. Babar, L. Chen, and K.-J. Stol, "A Systematic Review of Comparative Evidence of Aspect-Oriented Programming," *Information and Software Technology*, vol. 52, no. 9, pp. 871-887, 2010.
- [21] MIT Program Analysis Group. (2012, January) Daikon-related invariant detection publications. [Online]. <http://groups.csail.mit.edu/pag/daiikon/pubs/>
- [22] T. Kanstrén, *A Framework for Observation-Based Modelling in Model-Based Testing*. Oulu, Finland: VTT, 2010.
- [23] R. V. Binder, "Design for Testability in Object-Oriented Systems," *Communications of the ACM*, vol. 37, no. 9, pp. 87-101, Sept. 1994.
- [24] J. W. Nimmer and M. D. Ernst, "Invariant Inference for Static Checking: An Empirical Evaluation," *ACM SIGSOFT Software Engineering Notes*, vol. 27, no. 6, pp. 11-20, 2002.
- [25] L. Burdy et al., "An Overview of JML Tools and Applications," *Int'l. Journal in Software Tools for Technology Transfer*, vol. 7, no. 3, pp. 212-232, June 2005.
- [26] C. Csallner, N. Tillmann, and Y. Smaragdakis, "DySy: Dynamic Symbolic Execution for Invariant Inference," in *Int'l. Conf. on Software Eng. (ICSE 2008)*, Leipzig, Germany, 2008, pp. 281-290.
- [27] D. S. Rosenblum, "Towards a Method of Programming with Assertions," in *Int'l. Conf. on Software Eng. (ICSE 1992)*, Melbourne, Australia, 1992, pp. 92-104.
- [28] J. Yang, D. Evans, D. Bhardwaj, T. Bhat, and M. Das, "Perracotta: Mining Temporal API Rules from Imperfect Traces," in *Int'l. Conf. on Software Eng. (ICSE 2006)*, Shanghai, China, 2006, pp. 282-291.
- [29] B. Meyer, "Applying Design by Contract," *Computer*, vol. 25, no. 10, pp. 40-51, 1992.
- [30] S. Sims, R. Cleaveland, K. Butts, and S. Ranville, "Automated Validation of Software Models," in *Int'l. Conf. on Automated Software Eng. (ASE 2001)*, San Diego, USA, 2001, pp. 91-96.
- [31] S. McCamant and M. Ernst, "Early Identification of Incompatibilities in Multi-Component Upgrades," in *European Conf. on Object-Oriented Programming (ECOOP 2004)*, Oslo, Norway, 2004, pp. 440-464.
- [32] J. Whaley, M. C. Martin, and M. S. Lam, "Automatic Extraction of Object-Oriented Component Interfaces," in *Int'l. Symposium on Software Testing and Analysis (ISSTA 2002)*, Roma, Italy, 2002, pp. 218-228.
- [33] A. Coronato, A. d'Acierno, and G. De Pietro, "Automatic Implementation of Constraints in Component based Applications," *Information and Software Technology*, vol. 47, no. 7, pp. 497-509, 2005.
- [34] C. Csallner, Y. Smaragdakis, and T. Xie, "DSD-Crasher: A Hybrid Analysis Tool for Bug Finding," *ACM Transactions on Software Eng. and Methodology*, vol. 17, no. 2, pp. 1-37, 2008.

- [35] J-M. Jézéquel, D. Deveaux, and Y. Le Traon, "Reliable Objects: Lightweight Testing for OO Languages," *IEEE Software*, vol. 18, no. 4, pp. 76-83, 2001.
- [36] S. Thummalapenta and T. Xie, "Mining Exception-Handling Rules as Sequence Association Rules," in *Int'l. Conf. on Software Eng. (ICSE 2009)*, Vancouver, Canada, 2009, pp. 496-506.
- [37] J. Henkel, C. Reichenbach, and A. Diwan, "Discovering Documentation for Java Container Classes," *IEEE Transactions on Software Eng.*, vol. 33, no. 8, pp. 526-543, 2007.
- [38] C. Liu, L. Fei, X. Yan, J. Han, and S. P. Midkiff, "Statistical Debugging: A Hypothesis Testing-Based Approach," *IEEE Transactions on Software Eng.*, vol. 32, no. 10, pp. 831-848, Oct. 2006.
- [39] L. Mariani, S. Papagiannakis, and M. Pezzé, "Compatibility and Regression Testing of COTS-Component-Based Software," in *Int'l. Conf. on Software Eng. (ICSE 2007)*, Minneapolis, USA, 2007, pp. 85-95.
- [40] Y. Kataoka, M. Ernst, W. Griswold, and D. Notkin, "Automated Support for Program Refactoring Using Invariants," in *Int'l. Conf. on Software Maintenance (ICSM 2001)*, Florence, Italy, 2001, pp. 736-743.
- [41] N. Ubayashi, J. Piao, S. Shinotsuka, and T. Tamai, "Contract-Based Verification for Aspect-Oriented Programming," in *Int'l. Conf. on Software Testing, Verification, and Validation (ICST 2008)*, Lillehammer, Norway, 2008, pp. 180-189.
- [42] S. Ducasse, T. Gîrba, and R. Wuyts, "Object-Oriented Legacy System Trace-Based Logic Testing," in *European Conf. on Software Maintenance and Reeng. (CSMR 2006)*, Bari, Italy, 2006, pp. 37-46.
- [43] David Lo and Shahar Maoz, "Mining Scenario-Based Triggers and Effects," in *23rd Int'l. Conf. on Automated Software Engineering (ASE 2008)*, L'Aquila, Italy, 2008, pp. 109-118.
- [44] D. Ganesan et al., "Architectural Analysis of Systems based on the Publisher-Subscriber Style," in *Working Conference on Reverse Engineering (WCRE 2010)*, Boston, USA, 2010, pp. 173-182.
- [45] M. Christodorescu, S. Jha, and C. Kruegel, "Mining Specifications of Malicious Behaviour," in *Joint meeting of the European Software Eng. Conf. and the Symposium on the Foundations of Software Eng. (ESEC/FSE 2007)*, Dubrovnik, Croatia, 2007, pp. 5-14.
- [46] M. Cova, D. Balzarotti, V. Felmetser, and G. Vigna, "Swaddler: An Approach for the Anomaly-Based Detection of State Violations in Web Applications," in *Int'l. Symposium on Recent Advances in Intrusion Detection (RAID 2007)*, Queensland, Australia, 2007, pp. 63-86.
- [47] A. Baliga, V. Ganapathy, and L. Iftode, "Automatic Inference and Enforcement of Kernel Data Structure Invariants," in *24th Annual Computer Security Applications Conf. (ACSAC 2008)*, 2008, pp. 77-86.
- [48] D. Schuler, V. Dallmeier, and C. Lindig, "A Dynamic Birthmark for Java," in *Int'l. Conf. on Automated Software Eng. (ASE 2007)*, Atlanta, Georgia, USA, 2007, pp. 274-282.
- [49] M. Feng and R. Gupta, "Detecting Virus Mutations via Dynamic Matching," in *Int'l. Conf. on Software Maintenance (ICSM 2009)*, Edmonton, Canada, 2009, pp. 105-114.
- [50] M. Barnett and W. Schulte, "Runtime Verification of.NET Contracts," *Journal of Systems and Software*, vol. 65, no. 3, pp. 199-208, 2003.
- [51] Y. Le Traon, B. Baudry, and J-M. Jézéquel, "Design by Contract to Improve Software Vigilance," *IEEE Transactions on Software Eng.*, vol. 32, no. 8, pp. 571-586, Aug. 2006.
- [52] B. Demsky et al., "Inference and Enforcement of Data Structure Consistency Specifications," in *Int'l. Symposium on Software Testing and Analysis (ISSTA 2006)*, Portland, Maine, 2006, pp. 233-243.
- [53] D. Lorenzoli, L. Mariani, and M. Pezzé, "Towards Self-Protecting Enterprise Applications," in *Int'l. Symposium on Software Reliability (ISSRE 2007)*, Trollhättan, Sweden, 2007, pp. 39-48.
- [54] J. H. Perkins et al., "Automatically Patching Errors in Deployed Software," in *Symposium on Operating System Principles (SOSP 2009)*, Big Sky, USA, 2009, pp. 87-102.
- [55] Y. Wei et al., "Automatex Fixing of Programs with Contracts," in *Int'l. Symposium on Software Testing and Analysis (ISSTA 2010)*, Trento, Italy, 2010, pp. 61-71.
- [56] M. Book, T. Brückmann, V. Gruhn, and M. Hülder, "Specification and Control of Interface Responses to User Input in Rich Internet Applications," in *Int'l. Conf. on Automated Software Eng. (ASE 2009)*, Auckland, New Zealand, 2009, pp. 321-331.
- [57] J. Bumim and K. Sen, "Asserting and Checking Determinism for Multithreaded Programs," in *Joint meeting of the European Software Eng. Conf. and the Symposium on the Foundations of Software Eng. (ESEC/FSE 2009)*, Amsterdam, The Netherlands, 2009, pp. 3-12.
- [58] M. Gabel and Zhendong Su, "Online Inference and Enforcement of Temporal Properties," in *Int'l. Conf. on Software Eng. (ICSE 2010)*, Cape Town, South Africa, 2010, pp. 15-24.
- [59] C. Pacheso and M. D. Ernst, "Eclat: Automatic Generation and Classification of Test Inputs," in *European Conf. on Object-Oriented Programming (ECOOP 2005)*, Glasgow, UK, 2005, pp. 504-527.
- [60] J. H. Andrews and Y. Zhang, "General Test Result Checking with Log File Analysis," *IEEE Transaction on Software Eng.*, vol. 29, no. 7, pp. 634-648, July 2003.
- [61] A. M. Memon, "An Event-Flow Model of GUI-based Applications for Testing," *Journal of Software Testing, Verification and Reliability*, vol. 17, no. 3, pp. 137-157, 2007.
- [62] L. Ebrecht and K. Lemmer, "Highlighting the Essentials of the Behaviour of Reactive Systems in Test Descriptions Using the Behavioural Atomic Element," in *2nd Int'l. Conf. on Pervasive Patterns and Applications (PATTERNS 2010)*, Lisbon, Portugal, 2010, pp. 53-59.
- [63] M. Auguston, J. B. Michael, and M-T. Shin, "Environment Behavior Models for Automation of Testing and Assessment of System Safety," *Information and Software Technology*, vol. 48, no. 10, pp. 971-980, 2006.

- [64] M. Alshraideh and L. Bottaci, "Search-Based Software Test Data Generation for String Data Using Program-Specific Search Operators," *Software Testing, Verification and Reliability*, vol. 16, no. 3, pp. 175-203, 2006.
- [65] B. Korel and A. M. Al-Yami, "Assertion-Oriented Automated Test Data Generation," in *Int'l. Conf. on Software Eng. (ICSE 1996)*, Berlin, Germany, 1996, pp. 71-80.
- [66] T. Xie and D. Notkin, "Tool-Assisted Unit-Test Generation and Selection Based on Operational Abstractions," *Journal of Automated Software Eng.*, vol. 13, no. 3, pp. 345-371, July 2006.
- [67] M. Harder, J. Mellen, and M. D. Ernst, "Improving Test Suites via Operational Abstraction," in *Int'l. Conf. on Software Eng. (ICSE 2003)*, Portland, Oregon, USA, 2003, pp. 60-71.
- [68] J. E. Cook and A. L. Wolf, "Discovering Models of Software Processes from Event-Based Data," *ACM Transactions on Software Eng. and Methodology*, vol. 7, no. 3, pp. 215-249, 1998.
- [69] J. E. Cook and Z. Du, "Discovering Thread Interactions in a Concurrent System," *Journal of Systems and Software*, vol. 77, no. 3, pp. 285-297, Sept. 2005.
- [70] C. Csallner and Y. Smaragdakis, "Dynamically Discovering Likely Interface Invariants," in *Int'l. Conf. on Software Eng. (ICSE 2006)*, Shanghai, China, 2006.
- [71] J. Huselius and J. Andersson, "Model Synthesis for Real-Time Systems," in *9th European Conference on Software Maintenance and Reengineering (CSMR 2005)*, Manchester, UK, 2005, pp. 52-60.
- [72] S. Ali et al., "A State-Based Approach to Integration Testing based on UML Models," *Information and Software Technology*, vol. 49, no. 11-12, pp. 1087-1106, 2007.
- [73] R. Allen and D. Garlan, "Formalizing Architectural Connection," in *Int'l. Conf. on Software Eng. (ICSE 1994)*, Sorrento, Italy, 1994, pp. 71-80.
- [74] J. Yang and D. Evans, "Automatically Inferring Temporal Properties for Program Evolution," in *Int'l. Symposium on Software Reliability Eng. (ISSRE 2004)*, Saint-Malo, Bretagne, France, 2004, pp. 340-351.
- [75] N. Kuzmina, J. Paul, R. Gamboa, and J. Caldwell, "Extending Dynamic Constraint Detection with Disjunctive Constraints," in *Int'l. Workshop on Dynamic Analysis (WODA 2008)*, Seattle, Washington, 2008, pp. 57-63.
- [76] J. E. Cook and A. L. Wolf, "Event-Based Detection of Concurrency," in *6th International Symposium on Foundations of Software Engineering (FSE 1998)*, Paris, France, 1998, pp. 35-45.
- [77] M. Pradel and T. R. Gross, "Automatic Generation of Object Usage Specifications from Large Method Traces," in *Int'l. Conf. on Automated Software Eng. (ASE 2009)*, Auckland, New Zealand, 2009, pp. 371-382.
- [78] M. Gabel and Z. Su, "Javert: Fully Automatic Mining of Temporal Properties from Dynamic Traces," in *16th International Symposium on Foundations of Software Engineering (FSE 2008)*, Atlanta, USA, 2008, pp. 339-349.
- [79] D. Lo, G. Ramalingam, V. P. Ranganath, and K. Vaswani, "Mining Quantified Temporal Rules: Formalism, Algorithms, and Evaluation," in *Working Conference on Reverse Engineering (WCW 2009)*, Lille, France, 2009, pp. 62-71.
- [80] A. Memon and Q. Xie, "Using Transient/Persistent Errors to Develop Automated Test Oracles for Event-Driven Software," in *Int'l. Conf. on Automated Software Eng. (ASE 2004)*, Linz, Austria, 2004, pp. 186-195.
- [81] X. Deng, M. B. Dwyer, J. Hatcliff, and M. Mizuno, "Invariant-Based Specification, Synthesis and of Synchronization in Concurrent Programs," in *Int'l. Conf. on Software Eng. (ICSE 2002)*, Orlando, Florida, 2002, pp. 442-452.
- [82] A. Machetto, P. Tonella, and F. Ricca, "State-Based Testing of Ajax Web Application," in *Int'l. Conf. on Software Testing, Verification and Validation (ICST 2008)*, Lillehammer, Norway, 2008, pp. 121-130.
- [83] P. O. Meredith, D. Jin, F. Chen, and G. Rosu, "Efficient Monitoring of Parametric Context-Free Patterns," in *Int'l. Conf. on Automated Software Eng. (ASE 2008)*, L'Aquila, Italy, 2008, pp. 148-157.
- [84] N. Walkinshaw and K. Bogdanov, "Inferring Finite-State Models with Temporal Constraints," in *Int'l. Conf. on Automated Software Eng. (ASE 2008)*, L'Aquila, Italy, 2008, pp. 248-257.
- [85] S. Konrad and B. H.C. Cheng, "Real-Time Specification Patterns," in *Int'l. Conf. on Software Eng. (ICSE 2005)*, St. Louis, Missouri, USA, 2005, pp. 372-381.
- [86] C. Ackermann, M. Lindvall, and R. Cleaveland, "Recovering Views of Inter-System Interaction Behaviors," in *Working Conf. on Reverse Engineering (WCW 2009)*, Lille, France, 2009, pp. 53-61.
- [87] S. Thummalapenta and T. Xie, "Alattin: Mining Alternative Patterns for Detecting Neglected Conditions," in *Int'l. Conf. on Automated Software Eng. (ASE 2009)*, Auckland, New Zealand, 2009, pp. 283-294.
- [88] P. Bellini, P. Nesi, and D. Rogai, "Expressing and Organizing Real-Time Specification Patterns via Temporal Logics," *Journal of Systems and Software*, vol. 82, no. 2, pp. 183-196, 2009.
- [89] X. Yuan and A. M. Memon, "Iterative Execution-Feedback Model-Directed GUI Testing," *Information and Software Technology*, vol. 52, no. 5, pp. 559-575, 2010.
- [90] A. Wasylkowski and A. Zeller, "Mining Temporal Specifications from Object Usage," in *Int'l. Conf. on Automated Software Eng. (ASE 2009)*, Auckland, New Zealand, 2009, pp. 295-306.
- [91] K. Saleh, A. A. Boujarwah, and J. Al-Dallal, "Anomaly Detection in Concurrent Java Programs Using Dynamic Data Flow Analysis," *Information and Software Technology*, vol. 43, no. 15, pp. 973-981, 2001.
- [92] G. Marceau, G. H. Cooper, S. Krishnamurthi, and S. P. Reiss, "A Dataflow Language for Scriptable Debugging," in *Int'l. Conf. on Automated Software Engineering (ASE 2004)*, Linz, Austria, 2004, pp. 218-227.
- [93] A. Cavalli, C. Gervy, and S. Prokopenko, "New Approaches for Passive Testing Using an Extended Finite State Machine Specification," *Information and Software Technology*, vol.

- 45, no. 12, pp. 837-852, 2003.
- [94] P. J. Guo, J. H. Perkins, S. McCamant, and M. D. Ernst, "Dynamic Inference of Abstract Types," in *Int'l. Symposium on Software Testing and Analysis (ISSTA 2006)*, Portland, Maine, USA, 2006, pp. 255-265.
- [95] J. Henkel and A. Diwan, "Discovering Algebraic Specifications from Java Classes," in *European Conf. on Object-Oriented Programming (ECOOP 2003)*, Darmstadt, Germany, 2003, pp. 431-456.
- [96] N. Polikarpova, I. Ciupa, and B. Meyer, "A Comparative Study of Programmer-Written and Automatically Written Contracts," in *Int'l. Symposium on Software Testing and Analysis (ISSTA 2009)*, Chicago, USA, 2009, pp. 93-104.
- [97] Q. Xie and A. M. Memon, "Designing and Comparing Automated Test Oracles for GUI-Based Software Applications," *ACM Transactions of Software Eng. and Methodology*, vol. 16, no. 1, pp. 1-36, Feb. 2007.
- [98] N. Delgado, A. Q. Gates, and S. Roach, "A Taxonomy and Catalog of Runtime Software-Fault Monitoring Tools," *IEEE Transactions on Software Eng.*, vol. 12, no. 30, pp. 859-872, Dec. 2004.
- [99] C. Zhao, J. Kong, and K. Zhang, "Program Behavior Discovery and Verification: A Graph Grammar Approach," *IEEE Transactions on Software Eng.*, vol. 36, no. 3, pp. 431-448, May/June 2010.
- [100] Y. J. Ren and F. Chang, "ATTEST: A Testing Toolkit for Validating Software Properties," in *Int'l. Conf. on Software Maintenance (ICSM 2007)*, Paris, France, 2007, pp. 469-472.
- [101] T. H. Gibbs, B. A. Malloy, and J. F. Power, "Automated Validation of Class Invariants in C++ Applications," in *Int'l. Conf. on Automated Software Eng. (ASE 2002)*, Edinburgh, UK, 2002, pp. 205-214.
- [102] L. Froehrer, G. Glos, J. Osrael, and K. M. Goeschka, "Overview and Evaluation of Constraint Validation Approaches in Java," in *Int'l. Conf. on Software Eng. (ICSE 2007)*, Minneapolis, USA, 2007, pp. 313-322.
- [103] G. J. Holzmann, "The Logic of Bugs," in *Int'l. Symposium on the Foundations of Software Eng. (FSE 2002)*, Charleston, South Carolina, USA, 2002, pp. 81-87.
- [104] J. Burnim and K. Sen, "DETERMIN: Inferring Likely Deterministic Specifications of Multithreaded Programs," in *Int'l. Conf. on Software Eng. (ICSE 2010)*, Cape Town, South Africa, 2010, pp. 415-424.
- [105] A. Memon, I. Banerjee, and Adithya Nagarajan, "What Test Oracle Should I Use for Effective GUI Testing," in *Int'l. Conf. on Software Eng. (ICSE 2003)*, Portland, Oregon, USA, 2003, pp. 164-173.
- [106] T. Kanstrén. (2012, January) OSMOTester - Simple Model-Based Testing Tool. [Online]. <http://code.google.com/p/osmo/>

Interface Contracts for WCF Services with Code Contracts

Bernhard Hollunder

Department of Computer Science

Furtwangen University of Applied Sciences

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

Email: hollunder@hs-furtwangen.de

Abstract—Windows Communication Foundation (WCF) is a widely used technology for the creation and deployment of distributed services such as Web services. Code contracts are another .NET technology allowing the specification of pre-conditions, postconditions and invariants for .NET interfaces and classes. The embedded constraints are exploited for static analysis, runtime checking, and documentation generation. Basically, WCF services can be equipped with code contracts. However, it turns out that the WSDL interfaces generated for deployed WCF services do not include expressions for code contracts. Hence, the constraints imposed on WCF services are not visible for a service consumer. Though a proper integration of both technologies would bring additional expressive power to WCF and Web services, there does not exist a solution yet. In this paper, we present a novel approach that brings code contracts to WCF. Our solution comprises the integration of code contracts expressions at WSDL level as well as the generation of contracts aware proxies. The feasibility of the approach has been demonstrated by a proof of concept implementation.

Keywords-Code Contracts, Windows Communication Foundations, WCF, Web Services, WS-Policy, WSDL, Contracts Aware Proxies.

I. INTRODUCTION

Code contracts [2] are a specific realization of the *design by contract* concept proposed by Bertrand Meyer. With code contracts, i) methods of .NET types can be enhanced by preconditions and postconditions, and ii) .NET types can be equipped with invariant expressions that each instance of the type has to fulfill. While the application developer specifies code contracts for interfaces and classes, it is the responsibility of the runtime environment for checking the constraints and signaling violations. Furthermore, following tools are available for code contracts:

- Static code analysis;
- Documentation generation;
- Integration into VisualStudio IDE.

From a theoretical point of view, static code checking has its limitations and cannot detect all possible contract violations.

This is a revisited and substantially augmented version of “Code Contracts for Windows Communication Foundation (WCF)”, which appeared in the Proceedings of the Second International Conferences on Advanced Service Computing (Service Computation 2010) [1].

Nevertheless, it is a sophisticated instrument to help identifying common programming errors during compile time thus improving code quality at an early stage.

With the Windows Communication Foundation (WCF), service-oriented, distributed .NET applications can be developed and deployed on Windows. WCF provides a runtime environment for hosting services and enables the exposition of .NET types, i.e., Common Language Runtime (CLR) types, as distributed services. WCF employs well-known standards and specifications such as XML [3], WSDL [4], SOAP [5], and WS-Policy [6]. The Web Services Interoperability Technology (WSIT) project [7] demonstrates how to create Web services clients and implementations that interoperate between the Java platform and WCF.

When developing a WCF service, one starts with the definition of an interface (e.g., in C#) that is annotated with the `ServiceContract` attribute. Without this attribute, the interface would not be visible to a WCF client. To realize the service, a class is created that implements the interface. During service deployment, WCF will automatically generate an interface representation in the Web Services Description Language (WSDL) for the service. WSDL is programming language independent and allows the creation of client applications written in other programming languages (e.g., in Java) and running on different platforms. With the help of tools such as `svcutil.exe` and `wsdl2java` so-called proxy classes for specific programming languages can be generated. A proxy object takes a local service invocation and forwards the request to the real service implementation on server side by exchanging so-called SOAP documents.

In order to combine contracts with WCF, one may proceed as follows: The methods in a WCF service implementation class are equipped with code contracts expressions, that impose preconditions, postconditions, as well as object invariants. The C# compiler will not produce any errors and will create executable intermediate code, which can be deployed in a WCF environment. However, the constraints imposed by code contracts are completely ignored when WCF generates the WSDL for the service. As a consequence, a WCF client application cannot profit from the code contracts attached to the service implementation. This behavior has already been observed elsewhere [8]; however, a generic solution has not been elaborated yet.

This paper presents a novel approach for deriving interface contracts for WCF Services with code contracts. The strategy is as follows. When deploying a WCF service, the code contracts expressions contained in the service implementation class are extracted. Next, these expressions are encoded as assertions of WS-Policy [6]. The obtained WS-Policy description will be attached to the service's WSDL. On service consumer side, the generation of the proxy classes will be enhanced by including the code contracts expressions, which are extracted from the WS-Policy description of the WSDL.

The approach has the following features:

- It combines standard technologies such as WSDL and WS-Policy to bring code contracts to the interfaces of WCF services.
- The approach is transparent from a WCF service development point of view. The generation of both the interface contracts and the contracts aware proxy objects is completely automated.
- Code contracts will be already checked on client side, including static code analysis. This may save resources during runtime because invalid service requests will not be transmitted to server side.
- The feasibility of the approach has been demonstrated by a proof of concept implementation.

As indicated in [9, page 100], “software contracts play a key role in the design of classes for testability.” Thus, with our approach, a WCF developer has a further instrument for improving the quality of distributed .NET components. This is mainly due to the fact that additional constraints are now visible at the interface level in a formalized manner.

The paper is structured as follows. The next section will shortly introduce the underlying technologies. Section III will recapitulate the problem description; the solution proposed will be presented in Section IV. Section V will show how to represent code contracts with WS-Policy and how to attach a WS-Policy description to a WSDL file. Then, in Section VI, an implementation strategy (proof of concept) will be described. Section VII will give more details on interface contracts creation, followed by related work. Section IX concludes the paper.

II. FOUNDATIONS

This section will give a brief overview on the required technologies. We start with introducing code contracts, followed by WCF and WS-Policy.

A. Code Contracts

With code contracts [2] additional expressivity is brought to .NET interfaces and classes by means of preconditions, postconditions, and object invariants. A method can be equipped with preconditions and postconditions. A precondition is a contract on the state of the system when a method is invoked and typically imposes constraints on parameter values. Only if the precondition is satisfied, the

method is really executed; otherwise an exception is thrown. In contrast, a postcondition is evaluated when the method terminates, prior to exiting the method.

Code contracts provide a `Contract` class in the namespace `System.Diagnostics.Contracts`. Static methods of `Contract` are used to express preconditions and postconditions. To give an example, consider a method `squareRoot` that should not accept negative numbers. This could be encoded as follows:

```
using System.Diagnostics.Contracts;

class MyService {
    double squareRoot(double d) {
        Contract.Requires(d >= 0);
        Contract.Ensures(Contract.Result<int>() >= 0);
        return Math.Sqrt(d);
    }
}
```

Defining a precondition and a postcondition for `squareRoot`.

The `Contract.Requires` statement defines a precondition by means of a boolean expression. To specify the postcondition that the return value of `squareRoot` is also non-negative, we apply the `Contract.Ensures` method. With help of the expression `Contract.Result<int>` the return value of the method can be referred to.

Object invariants of code contracts are conditions that should hold on each instance of a class whenever that object is visible to a client. During runtime checking, invariants are checked at the end of each public method. In order to specify an invariant for a class, an extra method is introduced that is annotated with the attribute `ContractInvariantMethod`. Within this method, the conditions are defined with the method `Contract.Invariant`.

To give an example, consider the type `CustomerData` with members name, first name, identifier, and address. For sake of simplicity, the following excerpt does not show the complete class definition, but focuses on the specification of an invariant. The invariant ensures that any instance of `CustomerData` must have a name with a certain length, a non-negative identifier, as well as a real address instance.

```
using System.Diagnostics.Contracts;

class CustomerData {
    string name;
    string firstName;
    int identifier;
    Address address;

    [ContractInvariantMethod]
    void ObjectInvariant() {
        Contract.Invariant(
            name.length() >= 2 && identifier > 0 &&
            address != null);
    }
}
```

Definition of an invariant.

The expressions contained in code contracts 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. Code contracts also provide the universal quantifier `Contract.ForAll` and the existential quantifier `Contract.Exists`.

Both quantifiers expect a collection and a predicate, i.e., a unary method that returns a boolean. Universal quantification yields true if the predicate returns true on all the elements in the collection. Analogously, existential quantification checks whether the predicate is fulfilled for at least one element in the collection.

The above `squareRoot` example shows how preconditions and postconditions can be specified for *classes*. As a method in an *interface* is described only by its signature and does not have a body, `Contract.Requires` and `Contract.Ensures` statements cannot be part of an interface definition. Code contracts foresee a simple trick to encode constraints for interface methods: the required constraints are specified in a separate class, which is associated with the interface.

Suppose a class `AContract` should implement code contracts for an interface `IA`. Then `IA` is annotated with the attribute `[ContractClass(typeof(AContract))]`, and the class `AContract` is equipped with `[ContractClassFor(typeof(IA))]`. Now the code contracts of `AContract` apply to the interface `IA`.

Note that most methods of the `Contract` class are conditionally compiled. It can be configured via symbols to which degree code contracts should be applied during compilation. Code contracts can be completely turned on, which means that a full checking is performed, and off, i.e., all `Contract` methods are ignored. It is also possible to check only selected code contracts constraints such as preconditions (for details see [2]).

B. Windows Communication Foundation

According to [10], “WCF is a software development kit for developing and deploying services on Windows.” Services are autonomous, distributed and have well-defined interfaces.

An important feature of a WCF service is its location transparency: a consumer always uses a local proxy object – regardless of the location (local vs. remote) of the service implementation. The proxy object has the same interface as the service and forwards a call to the service implementation by exchanging SOAP documents. As the messages are independent of transport protocols, WCF services may communicate over different protocols such as HTTP, TCP, IPC and Web services.

The following listing shows the `squareRoot` functionality from above as a WCF service.

```
using System.ServiceModel;

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

public class ServiceImpl : IService {
    public double squareRoot(double d) {
        return Math.Sqrt(d);
    }
}
```

`squareRoot` as a WCF service.

The `ServiceContract` attribute maps the interface to a technology-neutral service contract in WSDL. To be part of the service contract, a method must be explicitly annotated with `OperationContract`. In order to implement the service, a class is created that inherits the interface as shown in the example.

Besides service contracts WCF also provides so-called data contracts. Data contracts are types, which can be passed to and from the service. There are built-in types such as `int` and `string`. Custom types can be declared as data contracts with help of the `DataContract` attribute. The `CustomerData` from above can be published as a data contract as follows:

```
using System.ServiceModel;

[DataContract]
class CustomerData {
    [DataMember]
    string name;
    [DataMember]
    string firstName;
    [DataMember]
    int identifier;
    [DataMember]
    Address address;
}
```

`CustomerData` as data contract.

In order to successfully deploy a WCF service, the WCF runtime environment requires the definition of at least one *endpoint*. An endpoint consists of

- an *address*,
- a *binding* defining a particular communication pattern,
- a *contract* that defines the exposed services.

Endpoints are typically defined in an XML configuration file (external to the service implementation), but can also be created programmatically.

During deployment, WCF generates a WSDL interface description for the service. A WSDL description has an interchangeable, XML-based format and comprises different parts, each addressing a specific topic such as the abstract interface and data types, the mapping onto a specific communication protocol such as HTTP, and the location of a specific WCF service implementation.

There are tools that transform WSDL descriptions into a programming language specific representation. Such a representation comprises classes for the proxy objects used by client applications. WCF delivers the tool `svcutil.exe`, which generates proxy classes for, e.g., C# together with a configuration file containing endpoint definitions. Basically, a proxy object constructs a SOAP message, which is sent to server side. A SOAP message consists of a body, containing the payload of the message (including the current parameter values of the request), and an optional header, containing additional information such as addressing or security data.

C. WS-Policy

When taking a closer look to a generated WSDL file one will find a couple of policy entries. These entries add further information to the service such as security requirements, reliable messaging, and arbitrary constraints. For example, it can be formally described that the parameter values of a request are to be encrypted during transmission.

WS-Policy is a widely used specification [6] to formulate policies in an interoperable manner. Almost all application servers including WCF support WS-Policy. In general, WS-Policy is a framework for defining policies, which comprise so-called (WS-Policy) assertions. A single assertion may represent a domain-specific capability, constraint or requirement and has an XML representation.

The following XML fragment shows how to associate a WS-Policy description to a service definition.

```
<definitions name="Service">
  <Policy wsu:Id="SamplePolicy">
    <ExactlyOne>
      <All>
        <IncludeTimestamp/>
        <EncryptedParts>
          <Body/>
        </EncryptedParts>
      </All>
    </ExactlyOne>
  </Policy>
  ...
  <binding name="IService" type="IService">
    <wsp:PolicyReference URI="#SamplePolicy"/>
    <operation name="squareRoot"> ... </operation>
  </binding>
  ...
</definitions>
```

WS-Policy attachment to a WSDL description.

In the example, a WS-Policy description is embedded into the WSDL of the `squareRoot` service. To be precise, via the `PolicyReference` element (for details see [6], [11]) a policy can be attached to a service. The top-level `Policy` element of a policy description has the child element `ExactlyOne`, which contains a set of so-called policy alternatives. Each alternative is surrounded by the `All` operator. The (single) alternative in the sample policy contains two assertions: `IncludeTimestamp` and

`EncryptedParts`. This policy requires that the caller of the service has to

- include a time stamp into the SOAP message
- encrypt the body of the SOAP message.

Note that an attached policy description is part of the WSDL interface of the service and must be taken into account by the service invoker. In the example, the WCF server side runtime environment would immediately reject the request (without performing `squareRoot`), if a client does neither include a time stamp nor encrypt the message body.

This example also demonstrates the declarative approach of WS-Policy. Additional, non-functional requirements can be formally described with corresponding assertions in a policy. Policies are external to the service implementation and can be simply combined. The WCF runtime environment has the responsibility to obey the policy.

WS-Policy itself does not come with concrete assertions. Instead, related specifications such as WS-SecurityPolicy [12] and WS-ReliableMessaging [13] apply WS-Policy to introduce specific assertions (e.g., `IncludeTimestamp` and `EncryptedParts` from the example above) covering specific domains. The respective specifications do not only define the syntax, but also the meaning of the assertions and their impact on the Web services runtime behavior.

WS-Policy has been designed in such a way that further, custom-designed assertions can be introduced. Our approach exploits this features to encode contracts expressions and to attach them to the service's WSDL.

III. PROBLEM DESCRIPTION

Suppose we want to create a WCF service with code contracts. A straightforward approach to combine both technologies would be as follows:

```
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);
    Contract.Ensures(Contract.Result<int>() >= 0);
    return Math.Sqrt(d);
  }
}
```

WCF service with code contracts.

We define a WCF service interface as usual according to the WCF programming model. In addition to the previous implementation of Section II-B, the `squareRoot` service is equipped with a precondition and a postcondition.

We first note that this WCF service implementation will be successfully compiled and deployed. However, we also observe that the WSDL description created during the deployment phase does not include any information about code contracts contained in the service's implementation. In other words, code contracts expressions are completely ignored and are not part of the WSDL interface.

There are two important consequences to stress here:

- 1) Code contracts imposed on the service implementation will not be considered when generating the proxy classes.
- 2) Clients of the WCF service are not aware of any code contracts expressions. Hence, code contracts support such as static analysis and runtime checking is not available on client side.

Thus, if a client invokes the `squareRoot` service passing a negative number as argument, the proxy object will forward the request to the server. The server itself will delegate the request to the service implementation. During the execution of the service, the code contracts runtime environment will eventually detect the violation of the precondition. The execution will be aborted and an exception will be returned to the client.

In the following, we will elaborate a concept that brings code contracts to the client side, thus enabling constraint checking already before passing the request via a network protocol to the server.

IV. CODE CONTRACTS AND WCF: THE CONCEPT

The overall concept of our solution for combining WCF and code contracts is illustrated in Figure 1.

One starts with implementing a WCF service according to both the WCF and the code contracts programming models. The service may use methods of the `Contract` class such as `Requires` and `Ensures` to specify preconditions and postconditions. For WCF data contracts, invariants can be defined.

During service deploying, the WCF infrastructure generates a WSDL for the service. Our approach will perform the following additional activities:

- 1) The code contracts expressions are extracted from the WCF service implementation class and are translated into corresponding WS-Policy assertions (so-called code contracts assertions).
- 2) A `PolicyReference` element is included into the WSDL of the service according to the WS-Policy-Attachment specification. The reference points to the code contracts assertions from the previous step.

The right part of Figure 1 shows the strategy to create proxy objects that are equipped with code contracts, so-called contracts aware proxies. In a first step, we derive the standard proxies from the WSDL by applying `svcutil.exe`

provided by WCF. Then, these proxies will be enhanced in the following way:

- 1) Extraction of the code contracts assertions attached to the service's WSDL.
- 2) Creation of corresponding preconditions as well as postconditions and their integration into the proxy classes. Classes derived for data contracts are extended by invariant methods.

Before we will discuss each of these steps in more detail, we make some observations. First of all, the standard programming models both for WCF and code contracts can be applied when implementing a service. The enhanced deployment infrastructure has the responsibility to perform the above mentioned activities. By automating these activities, our approach is transparent from a developer point of view.

Secondly, code contracts imposed on WCF services are also available for client technologies other than .NET. In fact, Web services technologies such as JAX-WS [14] extended by Java-based contract technologies can also profit from the code contracts assertions attached to the WCF service's WSDL. In Section VII-B, we will elaborate this feature in more detail.

Finally, we observe that our solution exploits and applies widely used technologies and specifications such as WS-Policy and WS-PolicyAttachment, which are supported by WCF and almost all Java-based Web services infrastructures. Thus, no proprietary frameworks must be installed to realize our approach.

V. CODE CONTRACTS ASSERTIONS FOR WS-POLICY

To formally represent code contracts expressions with WS-Policy, we introduce a WS-Policy assertion type, which is called `CodeContractsAssertion`.

The XML schema is defined as follows. Note that we omit, for sake of simplicity, some attributes such as `targetNamespace`.

```
<xsd:schema ...>
  <xsd:element name = "CodeContractsAssertion"/>
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name = "requires"
                    type = "xsd:string"
                    maxOccurs = "unbounded"/>
      <xsd:element name = "ensures"
                    type = "xsd:string"
                    maxOccurs = "unbounded"/>
      <xsd:element name = "invariant"
                    type = "xsd:string"
                    maxOccurs = "unbounded"/>
    </xsd:sequence>
    <xsd:attribute name = "context"
                  type = "xs:anyURI"
                  use = "required"/>
    <xsd:attribute name = "name"
                  type = "xs:anyURI"/>
  </xsd:complexType>
</xsd:schema>
```

XML schema for `CodeContractsAssertion`.

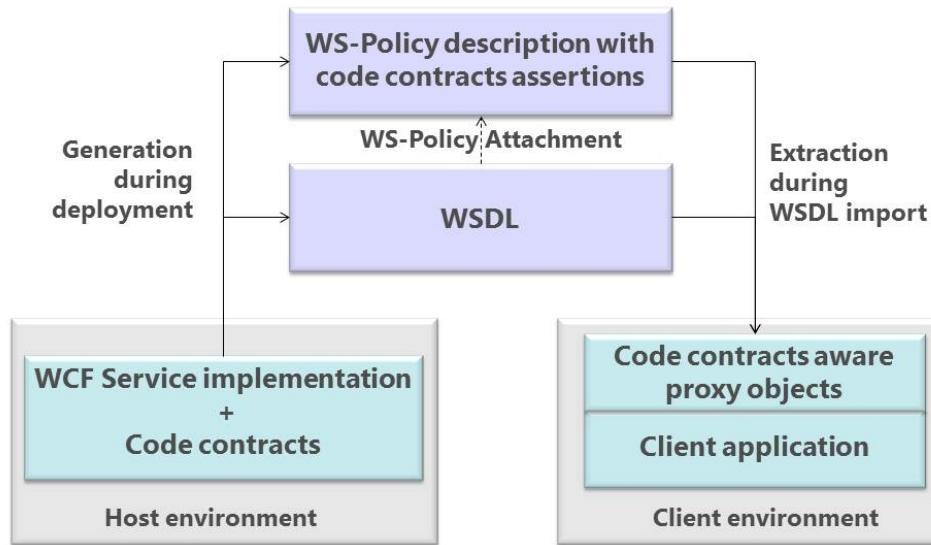


Figure 1. Combining code contracts and WCF.

A `CodeContractsAssertion` assertion has two attributes: a mandatory `context` and an optional `name`. The `context` attribute specifies the service to which the constraint applies. To be precise, the value of the `context` attribute is the (uniquely defined) name of a service as specified in the `binding` section of the WSDL. In case of an invariant, the `context` refers to a type defined in the `types` section. The `name` attribute can be applied to attach additional information to an assertion, which is not processed.

The body of a `CodeContractsAssertion` consists of a set of `requires`, `ensures`, and `invariant` elements. The values of these elements have the type `xsd:string` and should be valid code contracts expressions. An expression contained in a `requires` (resp. `ensures`) element represents a precondition (resp. postcondition) and typically refers to parameter names of the service. Note that these names are also part of the WSDL and can therefore be resolved properly.

An invariant expression applies to instances of data types used as service parameters. Such an expression may impose restrictions on the public members of the type, which are visible to a WCF client application.

Note that a code contracts expression contained in a service implementation class may impose restrictions on members, which are not visible – and hence are not meaningful – at WSDL interface level. Section VII will discuss this issue in more depth.

The created `CodeContractAssertions` are packaged into a WS-Policy description, which is attached via a `PolicyReference` to the service definition. The following WS-Policy description is produced for the WCF squareRoot service from Section III.

```

<definitions name="Service">
  <Policy wsu:Id="#CCPolicy">
    <ExactlyOne>
      <All>
        <CodeContractsAssertion
          name="squareRootAssertion"
          context=
            "IService.squareRoot(System.Double)">
          <requires>
            d >= 0
          </requires>
          <ensures>
            Contract.Result<int>() >= 0
          </ensures>
        </CodeContractsAssertion>
      </All>
    </ExactlyOne>
  </Policy>
  ...
  <binding name="IService" type="IService">
    <wsp:PolicyReference URI="#CCPolicy"/>
    <operation name="squareRoot"> ... </operation>
  </binding>
  ...
</definitions>

```

Code contracts policy for the squareRoot service.

In the following section, we will describe how to technically realize our concept.

VI. PROOF OF CONCEPT

We start with elaborating how to create and attach policies for code contracts during the WCF deployment process. Thereafter, we will focus on the activities at the service consumer side, which especially covers the generation of contracts aware proxies.

A. Code Contracts Extraction

Given a WCF service implementation, we need some mechanism to obtain its preconditions, postconditions and invariants. API functions have been published to access code contracts expressions. These functions are part of the *Common Compiler Infrastructure* project [15]. We adapted the proposed visitor pattern to obtain the methods' code contracts expressions and created a function `getCodeContractsForAssembly` that computes for a given assembly a code contracts dictionary. This dictionary is organized as follows:

- The *key* is the full qualified name of the method.
- The *value* is a list of strings each representing a code contracts expression. Each expression has the prefix `requires:`, `ensures:`, or `invariant:` to indicate its type.

The function `getCodeContractsForAssembly` can be implemented in straightforward manner by using types defined directly or indirectly in the namespace `Microsoft.Cci`.

B. Creation of WS-Policy Code Contracts Assertions

In this step, we create an XML representation for the code contracts expressions according to XML schema for `CodeContractsAssertion` as defined in the previous section.

We have realized a function `createCodeContractsAssertions` that takes a filled dictionary from the previous step. It iterates on the keys and performs the following actions:

- For each key, a `CodeContractsAssertion` is created. The value of its `context` attribute will be the key's name.
- For each expression contained in a key's value, an XML element is embedded into the `CodeContractsAssertion`. Depending on the well-defined prefix, the XML element will be either `requires`, `ensures`, or `invariant`.

The result of this step is a complete list of `CodeContractsAssertions` for the code contracts expressions in the service's implementation. How to embed a set of `CodeContractsAssertions` as a WS-Policy description into a WSDL file is described next.

C. WS-Policy Creation and Attachment

In WCF, additional policies can be attached to a WSDL file via custom bindings [16]. We define a custom binding that uses the `PolicyExporter` mechanism also provided by WCF. To achieve this, we implement two classes:

- `ExporterBindingElementConfigurationSection`
- `CCPolicyExporter`.

The former class is derived from the abstract WCF class `BindingElementExtensionElement`. The inherited method `CreateBindingElement` is implemented in such

a way that an instance of the `CCPolicyExporter` class is created. `CCPolicyExporter` has `BindingElement` as super class and implements the `ExportPolicy` method, which contains the specific logic for creating code contracts policies. Figure 2 visualizes the class layout.

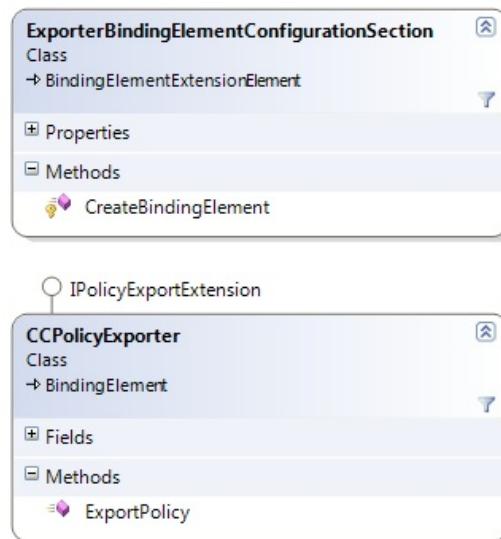


Figure 2. Class diagram for WS-Policy creation.

We have implemented the `ExportPolicy` method in the following way. In a first step, `getCodeContractsForAssembly` is invoked to obtain the filled dictionary. Next, `createCodeContractsAssertions` produces an XML representation for the code contracts expressions, which is then embedded into a valid WS-Policy description. Finally, a `policyReference` element pointing to this policy is embedded into the WSDL. Thus, we end up with an enriched WSDL description as shown at the end of the previous section.

To publish the service, a so-called endpoint must be configured (for details see, e.g., [10]). We have to change the standard configuration file of the WCF service such that the custom binding will be used:

- 1) In the definition of the service endpoint, the attribute `binding` is changed to `customBinding` and the attribute `bindingConfiguration` is set to `exporterBinding`.
- 2) In the `bindings` section, the `customBinding` element declares `exporterBinding`.
- 3) The element `bindingElementExtensions` is introduced in the `extensions` section. Its `add` element specifies the assembly in which the `ExporterBindingElementConfigurationSection` class is implemented.

During service deployment, WCF now uses the custom binding. As a result, the generated WSDL file will contain the code contracts policy.

D. Importing Code Contracts Policies

In order to invoke a service, a WCF client application requires a definition of a service endpoint. Typically, this is declared in a configuration file, similar to the one used on server side. In the `metadata` section of this file we included the definition of a so-called policy importer. By default, custom policies attached to a WSDL will not be evaluated when importing a WSDL.

In order to process code contracts policies, the policy-Importers element refers to the class CCPolicyImporter of Figure 3.

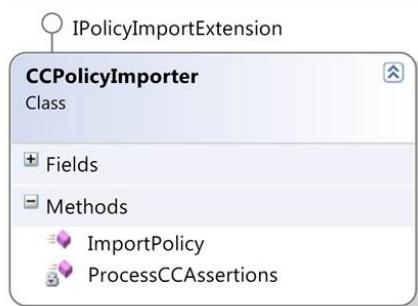


Figure 3. Class diagram for accessing WS-Policy descriptions.

We have realized this class in the following way. It implements the WCF interface `IPolicyImporterExtension`, which declares the `ImportPolicy` method. When processing a WSDL interface description, the runtime environment invokes this method and passes the attached policy description. `CCPolicyImporter` implements `ImportPolicy` in such a way that a code contracts dictionary (similar to the one on server side as described in Section VI-A) is constructed. To achieve this, the private method `Process-CCAssertions` iterates on the code contracts assertions of the policy and adds corresponding entries to the dictionary. This dictionary will be used to enhance the proxy classes, which is shown next.

E. Enhanced Proxy Generation

The tool `svccutil.exe` does not process custom policies. Hence, the standard proxy classes generated do not contain any code contracts constraints.

In our proof of concept we have realized the following approach. First, we apply `svctutil.exe` to create the standard proxy classes. In a second step, the following activities are performed:

- 1) Create an additional source file that will contain all constraints found in the code contracts policy. This file is called contract file.
 - 2) Link the contract file to the proxy class. Note that we do not want to modify the proxy class generated by `svcutil.exe`, because this would result in a strong dependency to the concrete code structure of the proxy.

Before we discuss the structure of the contract file, we observe that the generated proxy has a public interface (the proxy interface) that describes the supported services. There is also a public partial class (the proxy class) that implements the proxy interface. A client application instantiates the proxy class and invokes a provided service.

In the proof of concept, we apply the following strategy to bring the code contracts to the proxy. We create a new interface (the contract interface), that contains those methods that must be equipped with preconditions and post-conditions. We also introduce a further class (the contract class) implementing the contract interface. The inherited methods are implemented in the contract class in such a way that they contain the required `Contract.Requires` and `Contract.Ensures` statements. The contract class will be annotated with `ContractClassFor` to indicate that the constraints apply to the methods of the contract interface. The details for linking a contract class to an interface can be found at the end of Section II-A.

Next, we extend the partial proxy class in the contract file by inheriting the proxy interface. As the contract class is linked to the proxy interface, the preconditions and postconditions are also applicable to the proxy class.

For an invariant expression contained in the code contract policy we proceed as follows. The partial proxy class to which the invariant applies will be extended in the contract file by a new method that is annotated with `ContractInvariantMethod`. This method contains the required `Contract.Invariant` statements. This completes the generation and linkage of the contract file with the proxy generated by `svccutil.exe`.

F. Data Contracts

In WCF, data contracts are types that can be passed to and from the service. In addition to built-in types such as `int` and `string`, user defined data contracts can be introduced by annotating a class with the `DataContract` attribute. WCF will serialize all fields marked with `DataMember`. To impose object invariants on data contracts, methods annotated with `ContractInvariantMethod` will be introduced in the class context. The `Contract.Invariant` statements contained in these methods will then be managed by code contracts runtime.

As an example consider a data contract `CustomerData` (cf. Section II-A) with members such as name and address and an object invariant method that imposes restrictions on possible values. Suppose a WCF service `createCustomer` takes an instance of `CustomerData`. Because `CustomerData` is part of the service's signature, it has a representation as `complexType` in the WSDL. Therefore, `svcutil.exe` will generate a corresponding partial C# class `CustomerData`, which is used by the service consumer to construct instances. This class provides public setters

and getters for the members, but contains only a default constructor to create “empty” instances.

In order to invoke the `createCustomer` service, a client may proceed as follows:

- 1) Create an empty instance of `CustomerData`.
- 2) Set the specific values of the members with the public setters.
- 3) Pass the instance to the service.

Unfortunately, the code contracts runtime environment on client side will report an error after the first step. This is due to the fact that the empty members will (probably) define an invalid state of the instance, which is recognized by the object invariant.

To overcome this problem, one needs on client side a public constructor that takes all relevant customer data and constructs a properly initialized instance, which conforms to the object invariant. However, such a constructor is not generated by the standard `svcutil.exe` tool.

Therefore, part of the enhanced proxy generation is also the introduction of a suitable public constructor for a data contract class. This constructor will be part of the partial proxy class introduced in the contract file.

Observe that on WCF service provider side this is not an issue, though. When introducing a data contract, specific constructors can be implemented by the creator of the WCF service. These constructors are available for general usage on WCF provider side.

G. Exception Handling

There are two separated code contracts runtime environments: one on WCF service consumer side and one on WCF service provider side.

As described in Section 7 of [2], code contracts support several runtime behavior alternatives. By default, a contract violation yields an “assert on contract failure”. Thereafter, a user interaction is required to continue or abort program execution. While this behavior may be acceptable on client side during the development and testing phase, an analogous behavior would not be helpful on WCF provider side. Each time a violation occurs, the WCF service process requires a user interaction, which means that the server process must be observed the whole time. In general, this is not acceptable, not even during development and testing.

To remedy this problem, we disable “assert on contract failure” in the WCF service project. As a consequence, a contract violation now leads to the creation of an exception, which will be handled by the WCF runtime environment. By default, WCF returns a `FaultException` to the client indicating that something went wrong without giving detailed information. In order to embed the real reason into the exception (e.g., a “Precondition failed: $d \geq 0$ ” message) the `IncludeExceptionDetailInFaults` parameter of the `ServiceBehavior` attribute in the WCF service

implementation class is set to true as shown in the following listing:

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

[ServiceBehavior
(IncludeExceptionDetailInFaults = true)]
public class IServiceImpl : IService {
    public double squareRoot(double d) {
        Contract.Requires(d >= 0);
        Contract.Ensures(Contract.Result<int>() >= 0);
        return Math.Sqrt(d);
    }
}
```

Exception creation for a WCF service with code contracts.

On client side, standard exception handling can be applied to inspect the exception’s reason.

H. Development Model

To sum up, the development model that brings code contracts to WCF services is as follows:

- 1) Creation of a WCF service and an assembly with VisualStudio as usual, e.g., as *WCF Service Library* project.
- 2) Definition of a service endpoint that includes a modified configuration file with a custom binding as described in Section VI-C.
- 3) Deployment of the WCF service by launching the project. The published WSDL will contain a code contracts policy.
- 4) Creation of a WCF client project with VisualStudio as usual.
- 5) Invocation of the `ClientConnectorTool`, which is part of the proof of concept. This tool has a graphical user interface (see Figure 4) and generates for a specified WSDL contracts aware proxies, which will be included into a selected client project and assembly, respectively.
- 6) Usage of the code contracts infrastructure on client side.

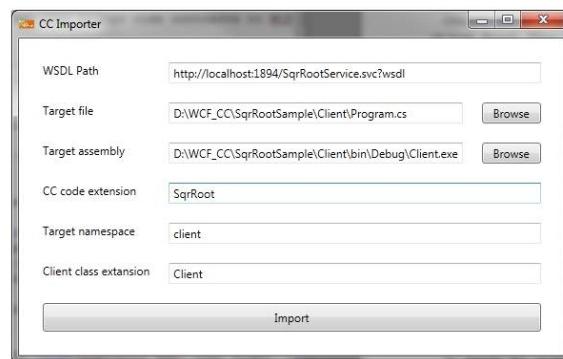


Figure 4. Tool for generating contracts aware proxies.

It should be noted that the code contracts processing is transparent to the developer – with the exception that the code contracts runtime environment and tools are now available on client side.

Our approach has the following advantages for the client developer. First, a static analysis of the code contracts can be performed, which helps detecting invalid invocations of WCF services during compile time. Second, during runtime a validation of the constraints will already be performed on client side. As a consequence, invalid service calls are not transmitted to the service implementation thus saving resources such as bandwidth and server consumption.

VII. CREATING CODE CONTRACTS POLICIES: A CLOSER LOOK

Before discussing interoperability issues in VII-B, we will observe that in general only a subset of the code contracts expressions of the service implementation should be mapped to code contracts policies.

A. Limitations

When specifying code contracts for a WCF implementation class, one may impose constraints on private members, which are not visible at the WSDL interface level. As a consequence, it is not helpful to map these constraints to contracts policies. Therefore, the generated contract policies should only contain constraints, which are meaningful to service consumers and hence can be validated on client side. To be precise, the code contracts policy should constrain the parameter and return values of WCF services as well as the public members of data contracts types. Constraints in the service implementation, which are not mapped to the code contracts policy, will be checked by the contracts environment on server side.

As mentioned in Section II-B, 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. For example, suppose that a precondition checks whether a parameter value of type `int` is a prime number. This can be achieved by invoking a custom predicate `isPrimeNumber` in the `Contract.Requires` statement. In order to check this precondition during service invocation, the implementation of the predicate must be available on client side. Thus, when importing code contracts policies on client side, only those predicates should be included into the contracts aware proxies, which are known on client side. Otherwise, the compiler will report an error on client side.

B. Interoperability

So far we have assumed that both on client and server side there is a .NET environment supporting the code contracts technology. Due to the interoperability of the Web services technology, Java based technologies may invoke WCF

services (see [14], Section 12). In our current approach, the code contracts expressions are implemented in a .NET language and must adhere to the specific syntax of C# or VB.

In order to use code contracts policies in a Java-based client environment, the expressions must be translated into equivalent Java expressions. As argued in [17], it is of advantage to represent expressions in contracts policies in a neutral, programming language independent format. In fact, the Object Constraint Language (OCL) of the Object Management Group [18] is a standardized language for formalizing constraints. In [17], the mappings from C# and Java, respectively, to OCL and vice versa are elaborated in more detail.

VIII. RELATED WORK

There are two research areas, which are related to our approach:

- Design by contract technologies;
- Constraints for policy languages.

While the former category comprises the realization of the *design by contract* principle for programming languages, the latter covers formalisms for specifying constraints for Web services.

Recently, several language extensions for Java have been proposed towards the specification of preconditions and postconditions for methods as well as invariants. Two well-known frameworks are *Contracts for Java* [19] and *Java Modeling Language* [20]. Typically, annotations such as `@Requires` and `@Ensures` are introduced by the frameworks to impose additional constraints. These approaches are targeting at the core Java programming language and do not address the impact on Web services environments such as JAX-WS. As in the case of WCF, these annotations are completely ignored when generating a service's WSDL. As argued in [17], our concept is rather generic and can also be applied to Java environments.

The formalization of non-functional requirements for Web services is a hot topic since the early days of Web services. Based on WS-Policy, WS-SecurityPolicy [12] is a well-known specification for imposing security constraints for Web services such as message integrity and confidentiality. There are proposals for defining domain-independent assertion languages such as WS-PolicyConstraints [21] and WS-Policy4MASC [22], which can in principle be used to encode code contracts expressions. However, these and related approaches do not address how to map constraints embedded in the service implementation to these formalisms.

IX. CONCLUSION

In this paper, we have elaborated a concept that brings code contracts to WCF. To be precise, we have shown how to i) derive interface contracts for WCF services and ii) create contracts aware proxy objects. As a consequence, WCF application developers can now profit from the additional

expressive power of code contracts including runtime and tool support. It has been stressed elsewhere that there did not exist a solution to the problem.

Our approach exploits well-known standards such as WSDL, WS-Policy, and WS-PolicyAttachment. We have described how to represent code contracts expressions by means of WS-Policy assertions. This representation will be used to generate an enhanced client proxy infrastructure, thus allowing the evaluation of the WCF service's code contracts already on client side. The developer of a WCF client application now explicitly sees important constraints imposed on the service implementation thus reducing the number of service invocations with invalid parameter values.

As noted in the section on related work, there are design by contracts approaches for Java. An interesting direction for future work is concerned with the question how interface contracts can be mapped to the Java environment. A solution may map code contracts expressions into a programming-language independent representation (e.g., in OCL). Afterwards, the OCL constraints will be translated into the specific syntax of the design by contracts technology used on service consumer side. Hence, a Java client application can also profit from the code contracts embedded in the WCF service implementation [17].

Currently, we are elaborating a tool that facilitates the development of Web services with Quality of Service (QoS) attributes such as security, performance and robustness [23]. As code contracts can be viewed as a sophisticated instrument to produce more robust and fault-tolerant software components, an interesting question is how to embed design by contracts technologies into the more general setting of tool support for arbitrary (QoS) attributes.

ACKNOWLEDGMENTS

I would like to thank the anonymous reviewers for giving helpful comments. This work has been partly supported by the German Ministry of Education and Research (BMBF) under research contract 17N0709.

REFERENCES

- [1] 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.
- [2] Microsoft Corporation, "Code contracts user manual," <http://research.microsoft.com/en-us/projects/contracts/userdoc.pdf>, last access Jan. 2012.
- [3] Extensible Markup Language (XML) 1.1. <http://www.w3.org/TR/xml11/>, last access Jan. 2012.
- [4] Web Services Description Language (WSDL) 1.1. <http://www.w3.org/TR/wsdl/>, last access Jan. 2012.
- [5] SOAP Version 1.2. <http://www.w3.org/TR/soap/>, last access Jan. 2012.
- [6] Web Services Policy 1.5 - Framework. <http://www.w3.org/TR/ws-policy/>, last access Jan. 2012.
- [7] Web Services Interoperability Technology (WSIT). <https://wsit.dev.java.net>, last access Jan. 2012.
- [8] Writing rock solid code with Code Contracts. <http://blog.hexadecimal.se/2009/3/9/>, last access Dec. 2011.
- [9] D. Esposito and A. Saltarello, *Microsoft .NET: Architecting Applications for the Enterprise*. Microsoft Press, 2009.
- [10] J. Löwy, *Programming WCF Services*. O'Reilly, 2007.
- [11] Web Services Policy 1.5 - Attachment. <http://www.w3.org/TR/ws-policy-attach/>, last access Jan. 2012.
- [12] WS-SecurityPolicy 1.3. <http://docs.oasis-open.org/ws-sx/wssecuritypolicy/v1.3>, last access Jan. 2012.
- [13] WS-ReliableMessaging 1.2. <http://docs.oasis-open.org/ws-rx/wsrn/v1.2/>, last access Jan. 2012
- [14] E. Hewitt, *Java SOA Cookbook*. O'Reilly, 2009.
- [15] Common Compiler Infrastructure: Code Model and AST API. <http://ccast.codeplex.com/>, last access Jan. 2012.
- [16] J. Smith, *Inside Windows Communication Foundation*. Microsoft Press, 2007.
- [17] B. Hollunder, "Deriving interface contracts for distributed services," in *Proceedings of the Third International Conferences on Advanced Service Computing (Service Computation 2011)*. Xpert Publishing Services, 2011.
- [18] OMG, "Object constraint language specification, version 2.2," <http://www.omg.org/spec/OCL/2.2>, last access Jan. 2012.
- [19] N. M. Le, "Contracts for java: A practical framework for contract programming," <http://code.google.com/p/cofoja/>, last access Jan. 2012.
- [20] Java Modeling Language. <http://www.jmlspecs.org/>, last access Jan. 2012.
- [21] 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.
- [22] 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.
- [23] B. Hollunder, A. Al-Moayed, and A. Wahl, "A tool chain for constructing QoS-aware web services," in *Performance and Dependability in Service Computing: Concepts, Techniques and Research Directions*. IGI Global, 2011.

Answering Complex Requests with Automatic Composition of Semantic Web Services

Brahim Batouche

*Public Research Center Henri Tudor,
Luxembourg
brahim.batouche@tudor.lu*

Yannick Naudet

*Public Research Center Henri Tudor,
Luxembourg
yannick.naudet@tudor.lu*

Frédéric Guinand

*University of Le Havre,
France
frédéric.guinand@univ-lehavre.fr*

Abstract—Today, there is a growing need for user to be able to express, and get answers to more complex requests, those including multiple functionalities, conditions, constraints and objectives. Complex requests including multiple functionalities only can not usually be answered with one single Web service.

As multiple services are needed, the problem is then to find good combination using the available services. This paper contributes to answering this issue. It focuses on the problem of semantic Web services composition to answer such requests.

We propose an automatic composition algorithm designing the answering composition. The algorithm takes into account all suitable composition structures. The set of answering composition is modeled as a graph, which supports the selection of the best composition according to the request constraints and objectives.

Keywords-complex web request; composition of web services; semantic web services.

I. INTRODUCTION

Electronic commerce (e-commerce) presents an important average gain for enterprises of different domains: tourism, transport, etc. For this reason the e-commerce has a growing interest in Web services to publish its products. Additionally, the Web services relative simplicity gives information providers and users easy access to new content. They are now widely available on the Web. For instance in 2009, the number of Web pages for e-tourism (electronic tourism) has been estimated to 65.2 billions, and represents in Europe 25.7% of the market [1]. As a side effect of this success, customer requirements become more and more complex, such that finding a single service fitting the specific needs of a user is unlikely.

Let us consider someone wanting to organize a stay in Rome. Using natural language, he/she could express his/her wish as: “ *I want to visit Rome for a week-end, I would like to go there by airplane and to stay in an hotel* ”. Such request has two functionalities: flight and hotel booking. A functionality is associated to a service and each one accepts inputs and produces outputs. The departure and destination cities of the flight are the input parameters of the flight service. The localization and the date are the input parameters of the hotel service. These parameters have to be

coherent, e.g. the arrival date in Rome has to be the same as the starting date of the Hotel reservation.

The request is processed in two steps: a first step for determining the services able to answer the different functionalities, i.e., the services allowing to book a flight and to book an hotel, and a second step for actually executing the functionalities, i.e., actual booking of flight and hotel. The first step is achieved by calling informative services (IS), while the second one is performed by active services (AS). IS output is used as input by AS.

This paper is an extension of [2]. We propose an algorithm for automatically finding all candidate compositions answering a complex request, without a priori knowledge of the composition structure. When the request does not formally specify any chaining between the request elements, the algorithm finds suitable composition structures based on the available services. The structure of the composite service depends obviously on the request, but also of the available services. The underlying problem is not trivial because there are many possible combinations of services, as well as many composition structures.

Services composition is useful in many different domains, e.g., tourism, transport, multimedia, etc. Some of them involve a dynamic environment where at any time events can affect previously computed compositions. The proposed algorithm can compensate services failures by finding a new executable composition when this happens.

In Section I-A and I-B we define respectively the research context and the useful definitions. In Section II, we present the state of the art and our contributions. In Section III, we formalize the problem elements: service, request and answering composition. In Section IV, we present how to describe semantically Web services and complex request. In Section V, we present the model of answering compositions. Section VI present our algorithm for automatic construction of a composition. Section VII is concerned with an experimentation results on which we assess our algorithm. Section VIII present specific cases of request resolutions. Finally, we conclude in Section IX.

A. Background

Complex requests such as the example presented here before, cannot be answered with one single service. Different processing steps are required. [3] decomposes the request resolution into: service presentation, service translation, composition generation, composition evaluation, and composition execution. In [4], the request processing is decomposed into: Web services discovery, planning, execution, and optimization. In [5], the request treatment follows three steps: discovery, plan generation and plan optimization. It appears that the choice of the resolution steps depends on the considered point of view: [4], [5] take the customer point of view, while [3] considers the provider point of view.

In this paper, we take the customer point of view and we consider both the composition design time and runtime to define our resolution steps. Design time can be decomposed in three steps: services discovery, composition design (or planning [4]) and selection of compositions. Runtime can be decomposed into execution and adaptation steps [6]. We propose a general resolution process for answering complex request made of five steps: (1) description of request functionalities, constraints and objectives; (2) services discovery: to find suitable services; (3) composition design: to determine how the services can be composed together to answer the request, and to design the corresponding composite service; (4) selection of answer(s): to select the composited services fulfilling the best the request objective(s) and respecting the request constraint(s), and finally (5) composition execution.

Figure 1 illustrates the different steps from the user request to the actual execution of the different services. The outputs of one step are the inputs of the following one. This paper focuses on steps (1) and (3), the step (2) being not considered as it is widely studied (see e.g., [7]). We also discuss the dynamic composition case, occurring when either services are faulty or when they can no more be invoked at execution time.

The request solving is decomposed into service discovery and service selection. When the request is complex we design the answer using composition of existing Web service. The automatic composition provides the answering composition set. Then the selection of the best composition can require an optimization method.

Automatic composition is based on automatic service discovery and combines the different discovered services to answer a complex request. Functional parameters in the request are used to find services, while non-functional parameters (request constraints and objectives) are exploited to select the best matching services.

B. Overview and Definitions

Web services composition consists in two steps: design time (design of composition answering a request) and runtime (composition execution). At design time, two steps can

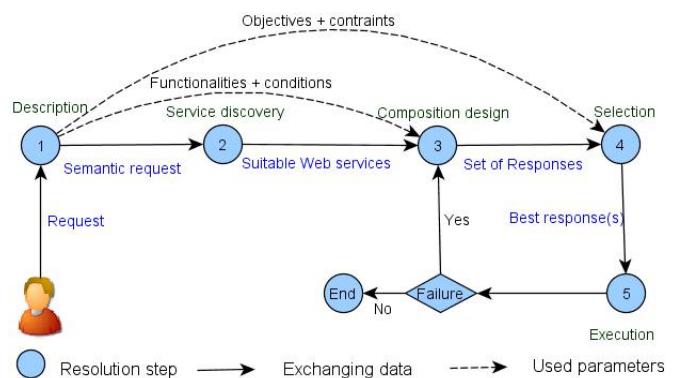


Figure 1. Complex request processing principle.

also be considered: (1) the search of possible compositions regarding the available services; and (2) the selection of best compositions based on request objective and constraint. In the following we provide related definitions.

1) *Composition Design*: The composition design can be performed in three ways: automatically, semi-automatically and manually. Moreover, the composition can be abstract or executable.

Manual composition is a combination of services directly specified directly by a designer. Semi-automatic composition determines the composing services and their control flow progressively by interrogating the user. Automatic composition determines the composing services and their control flow progressively without user interaction.

An executable composition allows services invocation and can be used as a composite service. An abstract composition comprises only the functionalities and their control flow, without giving any execution possibility.

Following our requirements, we consider executable services only. Indeed, the automatic composition requires being able to invoke some services at design time: in particular informative services, which will provide input data for other services. Additionally, the composition itself needs to be executable.

The composition of Web service is designed to achieve a specific goal. This goal is achieved by composing many services, and then building a new service, called “*composite service*”. The composite service can be modeled as a composition path, tree, organization of agents, chromosome, etc. The building of a composite service involves different composition structures such as: sequence, choice, switch, while, split (starting parallelism) and split-join (ending parallelism), see Figure 2. Many other terms are used for naming composition structures like: control flow, execution plan, and planning. In this paper these terms will be used as synonyms.

2) *Composition Runtime*: The composition runtime can be done in static or dynamic environment, the composition in dynamic environment is named “dynamic composition”

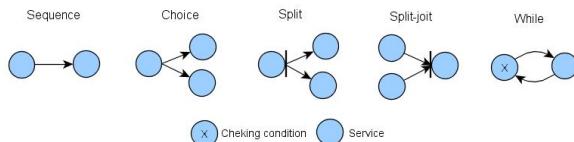


Figure 2. Composition structures illustration

while in static environment it is named “static composition”. The dynamic composition takes into account possible execution failures (service breakdown) and provides some executable alternative composition(s). This case is not considered in static composition [6].

II. STATE OF THE ART

Since our work concerns three items: complex request description, automatic composition design and answering composition model, this section presents the existing works in these areas and highlight our contributions.

A. Complex Request Description

Several works try to describe semantically the complex request. In [8], a complex request is presented as a set of inputs, outputs and conditions, which are formalized in description logic \mathcal{ALN} . The works [9], [10] present a complex request as an abstract composite Web service and use service profile of OWL-S [11]. The service model can be used to describe the internal structure. However, decomposing a request in a connected set of services, estimating data and control flow, is not obvious for users.

The request formalization used in [8], [10], [9] is somehow limited. [8] considers the request I/O and condition, but the use of \mathcal{ALN} as a formalization language makes it difficult to use with existing services standards implementations (using, e.g., OWL-S). The works [10], [9] use OWL-S description, but they do not consider all the request elements of formula 1, such as constraint and objective.

We modelled our request ontology to take into account all the elements specified in the request formalization (formula 1), and with the objective of keeping full advantage of existing works on semantic Web services, such as the matchmaking service OWL-MX [12]. Additionally, the user request needs to be formalized in such a way it can directly be used to find matching services.

B. Composition Models

Before analyzing the existing works, we present our requirements. According to our objectives, the composition model should fulfill the five following points: (1) represent a search space for optimization, specifying functional and non-functional parameters of each composition. (2) support all possible composition structures: sequence, choice, switch, while, split and split-join, (3) allow the composition invocation, (4) support dynamic environment and dynamic

composition: allowing to search for alternative compositions when failures occur during execution, (5) allow the translation of compositions into existing Web services description languages.

A model for the composition of Web services represents a set of services and their links, achieving a specific set of functionalities. According to researchers point of view and requirements, the composition is modeled by different mathematical representation: Petri nets [13], [14], [15], [16], [17], Directed Acyclic Graph [18], [19], [5], Workflows [20], [3], Situation Calculus [21], UML diagrams [22], Finite State System [23] or Multi Agents System (MAS) [24].

1) *Petri nets:* Are the widely used to model services composition, because they allow to model the steps and events in distributed system and consider the compositions structures. The composition is modeled with a set of places, transitions and tokens: places correspond to services, transitions correspond to input/output exchanges between services, tokens correspond to atomic operations [14]. Functional and non-functional parameters of services are specified in the places of Petri nets, and not specified in the transitions. We can not model services having values of parameters according to other service, because non-functional parameters are specified in the places. For example, when the hotel reservation cost changes according to the travel agency service, the hotel service can have different costs (non-functional parameter value), which cannot be specified in the same place [13], [14], [15], [16], [17]. Non-functional parameters could be specified as a transition weight vector, but Petri nets are not specifically designed to achieve this goal.

2) *Directed Acyclic Graphs:* DAG can be used to represent the execution order of services. The graph nodes represent the services and the arcs represent the data flow between services. The composition is modeled as a path. The functional parameters are specified in the node and the non-functional parameters are specified on the arcs. This structure allows to overcome the limit of Petri nets models. However, unlike Petri nets, DAG have some restrictions that prevent representing all the structures illustrated in Figure 2. For instance, DAG acyclic constraints is incompatible with the loop/while structure. To overcome this limit DAGs can be adapted for different composition structures [18], [19], [5].

3) *The following synthesis can be made about the other approaches:* Workflow presents a composition as a sequence of tasks and the message flow between services. The workflow semantic [20] allows to facilitate the interoperability of heterogeneous Web services. Workflows can model composition structures, but do not allow representing multiple compositions. *UML activity diagrams* is a descriptive model, inspired from Petri nets model [22]. The UML diagram is used for describing one compositions, and not a set. *Situation calculus* is efficient to construct the composition

in dynamic environment, because it represents the actions, and the situation as a sequence of actions. However, it is not obvious to exploit situation calculus to evaluate a composition. Additionally, it cannot be used to model multiple compositions, since it builds one composition step by step [21]. *Finite state system (FSS)* represents a set of states and transitions. States represent the services. Transitions consider all possible actions to execute a composition. However, FSS represents a composition as a sequence of services, and it is not well suited to consider other structures such as split or split-joint. Last, with *Multi Agents System (MAS)* [24] a services composition can be represented by a set of agents. Roles of agents corresponds to functionalities of services; input, output, precondition and effect. The MAS can be used to model the organization of compositions, but it provides a model that cannot be used as an optimization search space, because the non-functional parameters are not considered.

Each of the aforementioned models, has advantages and drawbacks for a specific requirement, but fails to meet all requirements listed at beginning of this section. Among the possible models we have listed, only the DAG is suitable, because it can be easily adapted to our requirements of supporting any composition structure and composition execution. Adaptation of the other models remains a priori too complex, without any foreseen benefit.

C. Automatic Composition Design

As defined in I-B1, automatic composition allows designing a composition without interrogating the user, defining the different Web services components and their composition structures. Some works [25], [26], [27], [15], [28], [29], [16], [24], [17] try to achieve this goal.

In [25], rules are used to generate a sequence of Web services, from the relations between them. Sequences are the only composition structures considered in that work.

In [26] a flooding algorithm is used. It first looks for services matching a request input. Then the algorithm progresses step by step by finding next services having input matching the output of previously selected services. The progression finishes when a selected service output matches with the request output.

An architecture for automatic Web service composition is proposed in [27], which according to the authors, allows fast composition of OWL-S services. The proposed approach uses implicitly a flooding algorithm. However, while authors provide interesting ideas for the design of the composite service and automating service invocation, they only consider sequences of services.

[15] proposes an algorithm based on matrix-equation approach and the provided compositions are modeled with Petri Nets. The compositions are built to answer a request. During the building, the method allows to check the reachability of the composition, by checking accessibility of states

from initial state. This approach is also useful to verify the validity of the built composition by other methods, but it does not allow the evaluation of the composition.

Oh, Lee, and Kumara [29] present an AI-planning algorithm of Web services composition, called WSPR. The algorithm relies on the request input and output, and then achieves the composition in two steps: (1) It computes the cost to achieve the composition, beginning with the request input. (2) This cost is used as guidance, to minimize the length of the sequence of services. This approach allows designing and selecting the composition at the same time, which it is not possible for any request, as discussed in section VIII-A. Additionally it considers only the sequence structure.

The algorithm in [28] builds a composition graph from a given request. It identifies first the input and output of the request and searches for a matching service. If none can be found, a service having only a matching output is selected and recursively, subsequent services having output matching with the input of the latter service and input matching with the request input are sought. The algorithm ends when a sequence of services starting with the request input and ending with its output is found, or when the set of available services has been visited. While the algorithm allows the composition execution, it is limited to sequences structures, such as [26]. The algorithm in [28] uses an inverse direction of built [26], and it has been designed to minimize the number of compositions, because generally an input can correspond to services having different outputs. The method of [26] is suitable to have a large set of compositions.

[16] presents an algorithm for “configuration” of compositions and selection of the best composition according to the services cost. The compositions are modeled with Petri Nets and are selected by considering the non-functional requirements. This approach allows to configure and select the composition at the same time, as in [29].

In [24], a MAS is used to answer a user request using automatic composition. The composition is based on a reasoning loop to determine the composition plans answering the request. An agent is limited to an OWL-S service and its functional parameters describing the agent role. The agents collaborate to provide the composition needed. The proposed reasoning algorithm allows to detect the composition plan according to the service semantic, but it does not consider all composition structures.

[17] builds the composition relying on semantic of service input/output. An hypergraph representation, as in [30], is used to determine the suitable services, and then to model the composition with Petri Nets. Using the semantic of services I/O ensures the coherency of composed services. The use of an hypergraph allows the determination of suitable compositions according to functional parameters, without considering non-functional parameters. However, it considers only the sequence structure. According to our requirements, it is

suitable to model all compositions without selection. The latter is reported in the compositions optimization step.

To automate the composition design, considering our requirements, the dedicated algorithm has to allow determining the composition structures and executing the composition. For this, we have to rely on the semantic of services input and output (as in [17]), and also on the semantic of composition structures.

D. Contributions

Our contributions to the Web services composition research field are respectively:

- 1) To formalize the different concepts of the problem: Web service, complex request and composition of Web service.
- 2) To propose an ontology (OWL-CR) describing semantically the complex request.
- 3) To model the composition by considering all requirement cited at beginning of section II-B.
- 4) To propose an algorithm solving a complex request with automatic composition, which considers any composition structures and detects them automatically.
- 5) To determine when the composition requires an optimization. When this is the case, we explain how to represent the composition set as a search space for optimization. In the other case, we explain how to select the best composition.

The existing studies address the composition problem only partially, and none of them considers all the points cited above. Additionally, no formalization of the problem can be found in the state of the art. In the following section, we clearly state the problem to solve.

III. FORMALIZATION

The problem space of automatic service composition concerns three main elements: a complex Web request to solve, a set of available Web services and service compositions that are form the services to answer the request. In this section, We formalize these elements.

A. Complex Request

Definition 1: A complex request illustrates a service a user asks for, following a specific goal, for which he/she specifies both functional and non-functional parameters. It can be represented as a set of functionalities on which conditions can be expressed, and a set of constraints and objectives expressing non-functional parameters.

The complex request can then be written as a three-tuple: $\mathcal{R} = \langle \mathcal{F}_{\mathcal{R}}, \mathcal{D}, \mathcal{N}\mathcal{F}_{\mathcal{R}} \rangle$, where $\mathcal{F}_{\mathcal{R}} = (F_{R_i})^T$ is the vector of functionalities, T being the transposition operator, where each functionality F_{R_i} is mandatory or optional and has as input and output respectively $I_{F_{R_i}}$ and $O_{F_{R_i}}$. $\mathcal{D} = \{d_1, \dots, d_r\}$ is a set of conditions, where a condition refers

to a request functionality input ($I_{F_{R_i}}$) or output ($O_{F_{R_i}}$); and $\mathcal{N}\mathcal{F}_{\mathcal{R}}$ is the set of non-functional parameters.

Lets first write $\mathcal{I}_{\mathcal{R}} = \{I_{F_{R_i}}\}$ and $\mathcal{O}_{\mathcal{R}} = \{O_{F_{R_i}}\}$, denoting respectively the input and output set of the request, $\mathcal{N}\mathcal{F}_{\mathcal{R}}$ can be defined as $\mathcal{N}\mathcal{F}_{\mathcal{R}} = \langle C, B, \lambda \rangle$, where $C = \{c_1, \dots, c_m\}$ is the set of constraints, $B = \{obj_1, \dots, obj_k\}$ is the set of objectives, and λ is the set of importance levels associated to objectives. Finally, this leads to the following formula:

$$\mathcal{R} = \langle \mathcal{I}_{\mathcal{R}}, \mathcal{O}_{\mathcal{R}}, \mathcal{D}, \mathcal{C}, \mathcal{B}, \lambda \rangle \quad (1)$$

Conditions \mathcal{D} differ from constraints \mathcal{C} in that they have to be verified by services answering a part of the request or for instantiating the input parameters of service (e.g., the depart and destination cities for a transport service), while constraints allow filtering the set of answering services or data obtained by the Informative Service (*IS*), or the composition obtained by the automatic composition process [2].

Objectives of \mathcal{B} will have to be minimized (e.g., cost, time) or maximized (e.g., performance, availability). In a specific context, an objective will be minimized (e.g., the service execution *time* is minimized) and maximized in another context (e.g., the leisure activity *time* is maximized).

Constraints and objectives can refer to data, e.g., the flight price must not exceed 200 Euro (C_d), or, minimize the flight price (B_d). They can refer to services, e.g., the service price must not exceed 5 Euro (C_s) (minimization); or to the composition, e.g., the travel price must not exceed 3000 Euro (C_c) (minimization). We write accordingly $\mathcal{C} = \langle C_d, C_s, C_c \rangle$ and $\mathcal{B} = \langle B_d, B_s, B_c \rangle$. It should be noticed that a composition constraint C_c can refer to one or multiple optional functionality(s) (e.g., if the price exceeds 2500 Euro, then cancel the sports activity service to reduce the price).

The importance levels λ are specified by the user. They can concern one single functionality (e.g., the transport price is more important than its quality.) or the whole request (e.g., the price is more important than the quality, globally). In the first case, we write $\lambda_{i,l}$, where i and l are respectively the objective and functionality indexes. In the latter case, it is simply written λ_i . The set of objectives importance levels can be written: $\lambda = \{\lambda_{i,l}, \lambda_i\}$. Additionally λ can be a single value (e.g., $\lambda_1 = 0.4, \lambda_2 = 0.6$) or an interval (e.g., $\lambda_1 \in [0, 0.4], \lambda_2 \in [0.6, 1]$). We have $\sum_{i=1}^{i=k} \lambda_i = 1$, and $\forall l, \sum_{i=1}^{i=k} \lambda_{i,l} = 1$. We note that a request functionality can be answered with a service or a composition of services. The latter can contain different types of services, informative, active or the both.

Finally, the request conditions can induce different kinds of dependency between functionalities. We distinguish three functionalities dependency types: no dependency, one-to-

one dependency and global dependency. The two latter are defined in Definition 2. Different strategies will be applied depending on their kind, as will be explained in section VIII-B.

Definition 2: One-to-one dependency in the request means that all dependencies existing between request functionalities concern at most two of them, and each functionality has at most one dependency.

Definition 3: Global dependency in the request means that there exists at least one functionality having more than one dependency.

B. Web Services

The literature proposes several definitions of Web services. Basically: *a Web service is a collection of protocols and standards used for exchanging data between applications* [31]. More specifically: *Web services are self-contained, modular business applications that have open, Internet-oriented, standards-based interfaces* [32]. Very specifically: *a Web service is a software application identified by a URI, whose interfaces and bindings are capable of being defined, described, and discovered as XML artifacts. A Web service supports direct interactions with other software agents using XML-based messages exchanged via Internet-based protocols* [33].

Mathematically, we formalize a Web service as a five-tuple:

$$\mathcal{S} := \langle \mathcal{ID}, \mathcal{FP}, \mathcal{NF}, \mathcal{CS}, \mathcal{NS} \rangle, \quad (2)$$

where \mathcal{ID} denotes the service identifier and access (e.g., service URI), \mathcal{FP} denotes its functional parameters, *Input, Output, Precondition, Effect* (IOPE); \mathcal{NF} denotes its non-functional parameters (e.g., service name, service price, etc.), \mathcal{CS} denotes the composition structure (i.e., the service control flow); and \mathcal{NS} is the set of ID inner services (or operations). The existing Web service languages try to annotate this five-tuple by describing them syntactically or semantically.

A Web service is modeled according to a specific goal. For this reason, there exist three Web service models [34]: black box, white box and semi-transparent box. The black box describes only the service functionalities, \mathcal{FP} , the semi-transparent box describes \mathcal{FP} and the service composition structures, \mathcal{CS} , and the white box details all inner services, \mathcal{NS} . According to [34], the white box is hardly useable, because it implies formalizing the service program in too much details.

Solving a complex request implies modeling both the existing services and the answering compositions. Regarding existing services, the black box model is privileged because knowing the functional parameter of services is enough to discover matching services. For the answering compositions,

which can be used as an optimization search space, the mixed use of white box and semi-transparent box (white-semitransparent box) is suitable. This allows describing all inner services \mathcal{NS} and composition structures \mathcal{CS} .

There are two types of service: informative service (*IS*) and active service (*AS*). The first provides some data (e.g., list of things) and does not modify its database after invocation. The second performs an action and modifies its database after its invocation (e.g., flight booking). There are significant differences between IS and AS regarding their usage in our resolution approach. Indeed, the IS will be executed at composition time to aggregate the provided information in the composition, whereas the AS will be executed at composition runtime, i.e., after the optimization has been performed, and after the user has selected its preferred composition among the best compositions proposed by the optimization method. The service can answer a request functionality partially or completely.

C. Composition of Web Services

Web services compositions is the combination of multiple service operations. These operations can follow a specific invocation order (i.e., a control flow or composition structure). A service composition can be formalized as a triple:

$$\mathcal{SC} := \langle \mathcal{OP}, \mathcal{CF}, \mathcal{E} \rangle, \quad (3)$$

where \mathcal{OP} denotes the set of service operations $\{op_1, \dots, op_j\}$, \mathcal{CF} denotes the set of control flow constructs (or composition structures) and $\mathcal{E} \subseteq (\mathcal{OP} \cup \mathcal{CF}) \times (\mathcal{OP} \cup \mathcal{CF})$ are edges connecting operations and control flow constructs [35]. For example, if we consider a DAG of composition $G = \langle N, A \rangle$, where N is the set of nodes and A the set of arcs, we have: $\mathcal{SC} := \langle \mathcal{OP} = N, \mathcal{CF} = \{Arc\ sequence\}, \mathcal{E} = A \rangle$.

In order to define the set \mathcal{CF} , we first review the most common Web service composition languages: OWL-S and BPEL4WS [36]. The existing languages for Web services composition allow to model different composition structures in different ways. Taking the most commonly known, we observed the following. The structures modeled by OWL-S are: "sequence", "any-order", "if-then-else", "choice", "while", "until", "split" and "split-joint". Differently, BPEL4WS [36] uses: "sequence", "switch", "while", "pick" and "flow". A mapping between the two representations involves two operators: equivalence (e.g., "choice" is equivalent to "pick"); decomposition (e.g., the "flow" structure in BPEL4WS can be decomposed into two structures of OWL-S: "split" and "split-joint"; "switch" is set of imbrication "if-then-else"). So, "while", "until", and "any-order" can be described by other structures.

In order to insure a compatibility with the different representations and keeping a generic approach, we focus

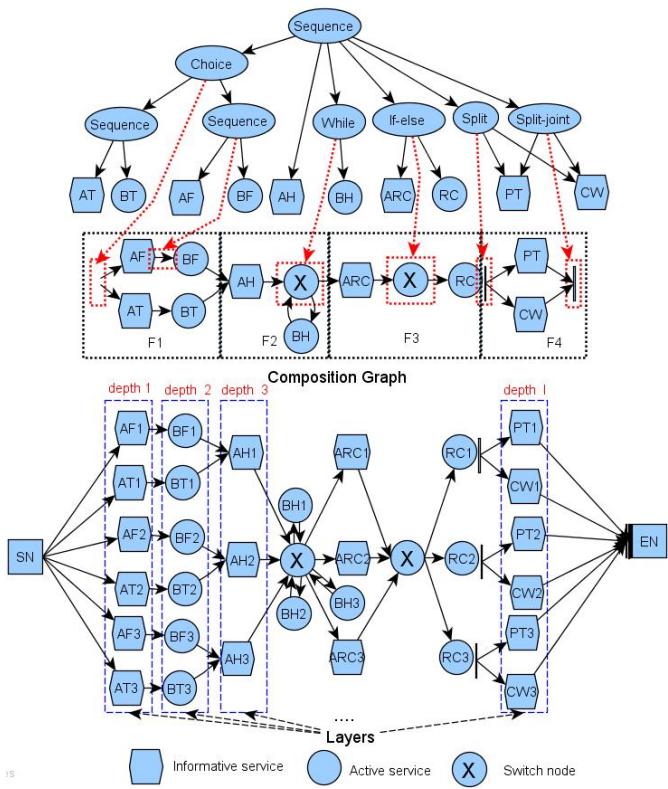


Figure 3. Service composition illustration: tree, flow and graph, where, AT means Available Train, AF means Available Flight, BT means Book Train, BF means Book Flight, AH means Available Hotel, BH means Book Hotel, ARC means Available Rentals Car, RC means Rent Car, PT means Plan Touristic map, CW means City Weather, SN start node and EN end node.

on elementary structures (sequence (*sq*), choice (*ch*), switch (*sw*), split (*sp*), split-joint (*sj*)), from which many others can be modeled. Consequently, \mathcal{CF} is defined as:

$$\mathcal{CF} = \{sq, ch, sw, sp, sj\}, \quad (4)$$

1) Composition Structures Illustration: A composition may comprise several different structures, which can themselves contain combinations of structures. A tree representation helps understanding and visualizing the composition: the leaves are service operations, the nodes and the root are the compositions structures. A flow representation corresponds to the reading of the composition tree by following a depth-first search.

Let us consider a composition answering the following request: “I want to travel from City A to City B, reserve several hotel rooms in destination city where each booking is billed separately, rent a car for six people, have the weather forecast and plan for the destination city”. The Figure 3 shows the composition tree, the corresponding composition path and the composition graph.

2) Characteristics of Composition Structures : In the following, we describe the different composition structures, and we provide a syntax for them.

Sequence ” \rightarrow ”: This structure defines a total order between services. There are two ways to detect the order: (1) checking the match between services IOPE (Input, output, precondition and effect); (2) checking the dependency between the services answering the question: *which service cancels the other when it is canceled?*

Choice ” $+$ ” (or-split): This structure represents a choice between several services that have a same functionality. $Choice(A, B_1, B_2, \dots, B_k) \equiv (A \rightarrow B_1) \vee (A \rightarrow B_2) \vee \dots \vee (A \rightarrow B_k)$, knowing that service *A* precedes all services $\{B_i\}$ which have the same functionality.

If-then-else ” \otimes_c ”: This structure is the classical condition branching. It allows a conditional service execution or choice of services input parameters values. This structure checks a request condition, and controls a service if at least one of its functional parameters corresponds to a request condition predicate. The switch structure is based on If-then-else structure, because it represents an imbrication of this latter. In the following, switch and if-then-else terms are used as synonyms.

Split ” \vdash ”: This structure indicates a simultaneous start of multiple services (or services chains) that can be parallelized. The parallelized services have the same predecessor and provide different types of outputs. Each parallelized service can start a new sub-path in the composition. All services chains starting at a split will be executed in parallel and ended with a split-joint. $Split(A, B_1, B_2, \dots, B_k) \equiv (A \rightarrow B_1) \wedge (A \rightarrow B_2) \wedge \dots \wedge (A \rightarrow B_k)$.

Split-joint ” \dashv ”: This structure ends a parallel structure, where different composition paths belong to a same ”split” and the last services $\{B_i\}$, in parallel chains, have the same successor *A*. $Split-Joint(B_1, B_2, \dots, B_k, A) \equiv (B_1 \rightarrow A) \wedge (B_2 \rightarrow A) \wedge \dots \wedge (B_k \rightarrow A)$, where services $\{B_i\}$ end the parallel composition paths. It is possible that services from a same split do not end with the same Split-joint.

Any-Order ” \odot ”: This structure represents a random invocation of services. It is not elementary because it means the choice between all possible alternatives, i.e., it can be expressed using choice and sequence structures: $A \odot B \equiv (A \rightarrow B) + (B \rightarrow A)$. At execution time, such structure can be replaced by a sequence or a parallel structure [37]. It can be noticed that if the number of services involved in the any-order structure is large, the replacement by a parallel structure may be very costly, since it is a combinatorial enumeration.

While ” \oplus_c ” and until ” \ominus_c ”: These structures are used for iterative service invocation, and they are not elementary because they can be constructed with if-then-else and sequence structures: $\oplus_c(A) = \otimes_c \rightarrow A \wedge A \rightarrow \otimes_c$ and $\ominus_c(A) = A \rightarrow \otimes_c \wedge \otimes_c \rightarrow A$.

The Table I summarizes the detection rules of composition

Composition structure	Detection rule
Sequence (A, B)	$M(O_A, I_B) \leq \epsilon \vee (M(O_A, O_{\mathcal{R}}[i]) \leq \epsilon \wedge M(\mathcal{I}_{\mathcal{R}}[i+1], I_B) \leq \epsilon)$
Choice (B_i, B_j)	$M(O_{B_i}, O_{B_j}) \leq \epsilon \wedge A \rightarrow \{B_i, B_j\}$
If-then-else (A)	$dom(a_i) = O_A$
Split (A, B_i, B_j)	$M(O_{B_i}, O_{B_j}) > \epsilon \wedge A \rightarrow \{B_i, B_j\}$
Split-join(B_i, B_j, A)	$M(O_{B_i}, O_{B_j}) \leq \epsilon \wedge \{B_i, B_j\} \rightarrow A \wedge n.id(B_i) \equiv n.id(B_j)$

Table I
DETECTION RULE OF ELEMENTARY COMPOSITION STRUCTURES,

where, I, O are respectively input,output; A,B are Web service; $\mathcal{I}_{\mathcal{R}}, \mathcal{O}_{\mathcal{R}}$ are respectively the request input/output; a_i is a predicate of request condition ; $n.id(B_i)$ denotes the split identification, where service B_i belong, M the matching level and ϵ the matching threshold.

structures.

In our example in Figure 3, the AF and AT are structured with “choice” structure, because their output classes match. The AH service is controlled with “switch” structure because the request condition “*rent a car for six people*” refers to the car capacity. The latter is a property of the ARC output class. So on for the other structures.

IV. SEMANTIC DESCRIPTION

The complex request resolution requires describing semantically the Web services and the complex request. Semantic web services provides knowledge to discover, compose and invocate services. The semantic description of a complex request provides knowledge needed to solve the request.

A. Semantic Web Service

Semantic Web Services approaches add a semantic layer to elements of Web services, $\mathcal{ID}, \mathcal{FP}, \mathcal{NF}, \mathcal{CS}, \mathcal{NS}$ (see equation 2). This allows the automation of discovery, composition and invocation of Web services over the Web. Then, the main question is: *Which semantic layer is needed to automate service discovery, composition and invocation?*.

Existing works focus on specific elements of equation 2. For example, the WSLA (Web Service Level Agreement) project [38] adds a semantic layer on \mathcal{NF} , addressing service management issues and challenges in a Web services environment on SLA specification, creation and monitoring. The WSDL-S [39] language adds a semantic layer to WSDL, i.e., on $\mathcal{ID}, \mathcal{CS}$ and \mathcal{FP} . The METEOR-S (Managing End-To-End Operations for Semantic Web Services) project [40] presents a semantic annotation of WSDL, addressing the issues related to workflow process management for large-scale, complex workflow process applications in real-world multi-enterprise heterogeneous computing environments. METEOR-S is based on approaches described in [41], [18], to define an extensible ontology that describes the properties of quality of service.

According to our point of view, services discovery can be automated by adding a semantic layer to \mathcal{FP} ; services composition can be automated by adding a semantic layer to $\mathcal{FP}, \mathcal{CS}$ and \mathcal{NS} ; and service invocation can be automated by adding a semantic layer to \mathcal{ID} and \mathcal{FP} . OWL-S is

currently the most known and referred language for semantic web services. It allows more flexibility than METEOR-S by relying on a domain ontology. The process view taken by the latter not only induces some complexity in the reasoning, but also is unnecessary for the needs of our research.

The Semantic Web Services language OWL-S provides semantic layers namely for \mathcal{ID} (service grounding), \mathcal{CS} and \mathcal{NS} (service model). \mathcal{FP} is modeled by the service profile, but the semantics of IO is left to the additional use of a domain ontology, as well as the semantics of PE, which in [10] is described with SWRL rules. \mathcal{NF} description is also left to the use of an additional QoS ontology.

In the following section, we present an ontology for describing complex requests and useable with OWL-S services description.

B. The Complex Request Ontology: OWL-CR

The ontology for Complex Request we have designed is illustrated in Figure 4. The root class of OWL-CR is the class Request. Two main properties, namely *hasFunctionality* and *hasNFparameter*, allow respectively to formalize the different functionalities and non-functional parameters of a request.

For each functionality, a Functionality Profile is defined, which can be Mandatory or Optional. The optional profile is not taken into account when its consideration induces the non-respect of a constraint. Each element of the request in formula 1 has a corresponding class in the ontology. For a given request \mathcal{R} , the $I_{F_{R_i}}, O_{F_{R_i}}$ are input or output classes of a service domain ontology (in our case, the travel booking service ontology), the condition \mathcal{D} is described using the `ruleml:Impl` class of SWRL, defining a rule. This is formalized using respectively the *hasInput*, *hasOutput* and *hasCondition* properties. The parameter type in Input and Output classes allow specifying the URI of corresponding classes in a service domain ontology modeling services characteristics in a particular domain.

The non-functional parameters, constraints \mathcal{C} and the objectives \mathcal{B} , are respectively instances of Constraint and Objective, which can depend on the non-functional parameters of Service, Data or Answered composition. The importance level of an objective λ

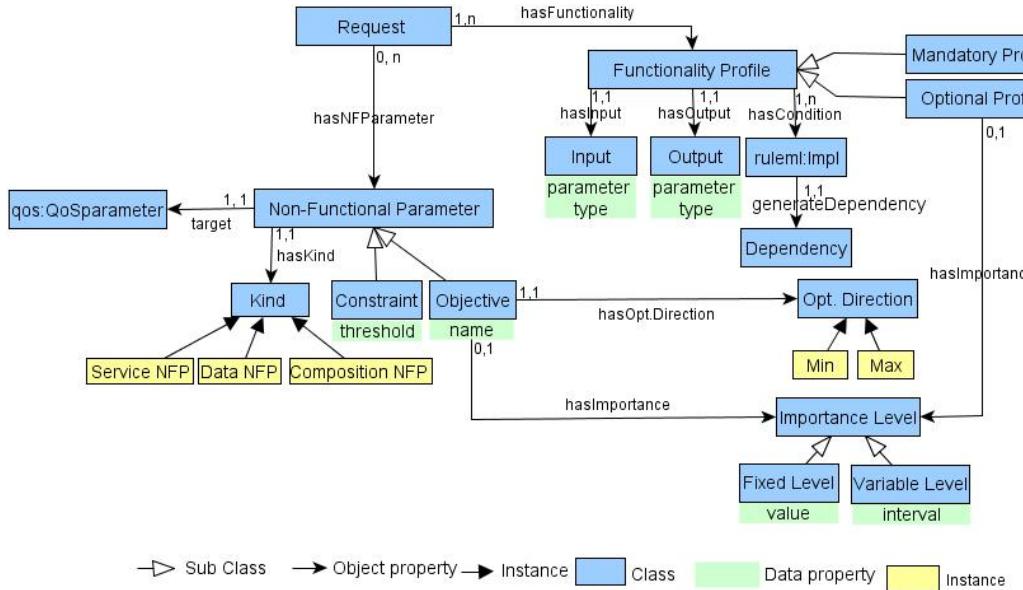


Figure 4. Web service complex request ontology (OWL-CR)

is an instance of Importance Level, which can be a Fixed level or Variable level. The objective has an optimization direction: Min or Max, used to specify how it has to be exploited.

The ontology OWL-CR contains different cardinalities (0,1), (0, n), (1,1) and (1,n), where $n \geq 1$. According to the OWL language capability, it has been formalized in OWL DL. OWL-CR is used as a language to describe complex requests. Practically, this means that users inputs, through a dedicated user interface, are used to construct a request description formalized in OWL-CR.

For an ideal situation, functional and non-functional parameters of the request need to be expressed using the same domain and QoS ontologies of the services to be queried. The OWL-CR ontology we have designed for expressing requests uses directly these ontologies. Moreover, it exploits SWRL to formalize the conditions.

V. MODELS OF ANSWERS

The goal of this section is to present how the set of valid compositions answering a request is modeled, according to the request characteristics. The request functionality dependencies have an impact on the adapted compositions models. One can numerate three cases:

- 1) When no dependency is specified, the composition is modeled as a sequence following a random order and the set of compositions can be modeled as a graph.
- 2) In the case of one-to-one dependencies, the composition is modeled as a sequence following the specified order of dependent functionalities and the set of composition can be modeled as a graph.

- 3) When the dependencies are global, a composition can be modeled as a sequence and we cannot model the set of valid compositions as a graph. We detail this point in section VIII-B.

A. Graph of Compositions

When the request contains no dependency or one-to-one dependencies, the compositions answering the request are modeled as a graph. The composition graph G models the set of possible connexion between services. The complete paths starting from the first node and ending at the last node of the graph, are composition answering the request. G is composed of several layers according to request functionalities. Each layer contains a set of services answering the same request functionality. The composition graph is formulated as: $G = \langle N, A \rangle$, where N is a set of nodes $\{n_j\}$ and A is a set of arcs $\{a_i\}$. An arc is defined as $a_i = \langle b, e, w \rangle$, where b is the depart node, e is the destination node and w is a weight. Nodes n_i correspond to service. G has two special nodes representing the start, $sn = \langle I_{F_{R_1}} \rangle$, where $I_{F_{R_1}}$ is the first element of the request input; and the end of all compositions $en = \langle O_{F_{R_l}} \rangle$, where l is the index of the last functionality. The composition paths of G are defined by the following definitions.

Definition 4: Let Ω be the set of paths in a composition graph answering a request R . We have:

$$\forall x = (b \rightarrow e) \in \Omega : b = sn, e = en$$

Corollary 1: $\forall x \in \Omega, \forall c \in C_c \cup C_d : x \models c, \exists I, M(I_{F_{R_i}}, I) \leq \epsilon, \exists O, M(O_{F_{R_j}}, O) \leq \epsilon, \forall I_{F_{R_i}} \in \mathcal{I}_R, \forall O_{F_{R_j}} \in \mathcal{O}_R$.

Where \models is the satisfaction symbol, I/O is the input/output of inner services of x , C_c, C_d are set of con-

straints referring respectively to composition and data, ϵ is the matching threshold level and $M(y, z)$ is the matching level between y and z .

A composition path has both functional and non-functional parameters. Functional parameters are affected to the nodes (i.e., service I/O), and non-functional parameters are affected to arcs. An arc weight is a vector containing request objectives values v_{obj_i} for the arc destination node. Formally, let $a_{i,j} : (n_i \rightarrow n_j)$ be an arc and $w_{a_{i,j}}$ its weight, $w_{a_{i,j}} = (v_{obj_1}(n_j), \dots, v_{obj_k}(n_j))^T$.

The composition graph is executable when it allows invoking the compositions. The nodes then contain the communication protocol, which is specified in service grounding of OWL-S description.

The composition graph can consider all defined composition structures or only the sequence structure. Considering all composition structures is a way to expand the search space of objectives, because we consider different evaluation functions of an objective.

The evaluation function of objectives can change according to the composition structure (as detailed in table ??). For example, the duration (dr) objective has different evaluation functions: ($\max(dr_i)$) for a parallel structure and ($\sum dr_i$) for a sequence structure. For some parameters, it never changes. For example, the price (pr) objective has the evaluation function ($\sum pr_i$), for any composition structures.

When the evaluation functions of all request objectives are the same for any composition structures, we consider only the sequence structure and the composition graph is then acyclic. The latter can become cyclic if we consider additionally if-then-else structures. When we consider all possible composition structures, the obtained composition graph is called multi-structure: its paths are adapted to consider the composition structures.

In summary, considering the request functionalities have no global dependency, when the request objectives have the same evaluation function f_{obj_i} for all the composition structures (cs), the composition graph is acyclic. Otherwise, the graph is multi-structures.

1) Multi-Structures Composition Graph: In order to answer all of our requirements and in particular considering all compositions structures. We adapt the DAG to a Multi-Structures Composition Graph (MSCG). We define a node as a six-tuple:

$$n = \langle NT, id, URI_S, URI_{DA}, I_s, O_s \rangle, \quad (5)$$

where NT is the node type, with $NT = \{IS, AS, DA, SW\}$, where IS is an informative service, AS is an active service, DA is data and SW is a switch node that represents a conditional structure. id is an identifier for nodes starting a parallel structure ($id = \phi$ if n does not belong to a parallel structure); URI_S is the service URI ($URI_S = \phi$ if $NT =$

DA); and URI_{DA} is the data URI ($URI_{DA} = \phi$ if $NT = IS$ or AS). I_s/O_s are respectively the input and output of the service represented by the node.

The formalization (5) is suitable to have short runtime complexity but unsuitable to have low memory complexity, because some elements are useful only during the composition design, such as NT, I_s, O_s . However, the nodes are memorized as : $n = \langle URI, id \rangle$.

The switch node is used to model the request conditions D in each composition. It is defined as: $n_{SW} = \langle NT = SW, d_i, HF \rangle$, where d_i is a request condition, and HF is a hash function, that is used to store reference to nodes verifying the condition. For this reason, the switch node is a kind of meta-node containing several nodes.

In the MSCG, composition structures (St) are represented as a combination of a node and arcs. We write: $St := \langle N, A \rangle$. The Table II gives the different forms for each kind of structure.

Structures	node definition
while and until	$\langle SW, A_1 \rangle$
if-then-else	$\langle SW, \phi \rangle$
sequence, choice, split and split-joint	$\langle \phi, A_2 \rangle$

Table II
DEFINITION GRAPHIC OF COMPOSITION STRUCTURES,

where A_1 is the sequence-arcs, and A_2 is respectively the set of sequence-arc, split-arc and split-joint-arc.

In MSCG, the arc weights must be adapted for specific cases. If the destination node type is data ($NT(n_j) = DA$), then some objectives are not defined, because non-functional parameter of service are not the same at data base property. For example, the runtime objective might be missing in data representing hotel related information. In this case, we affect a neutral value to the objective. If the type of the arc destination node is “switch” ($NT(n_j) = sw$), then $w_{a_{i,j}}$ will be affected by neutral value, because switch node has any impact in weight of composition.

As detailed in section VIII-A, the optimization can be required. If so, the MSCG is transformed to provide a suitable search space. The composition structures (if-then-else, parallel) have not any impact on optimization and are note considered. The MSCG will then be transformed by reducing these structures. The obtained composition graph is cyclic if the MSCG contains while/until structures, it is acyclic otherwise.

The split and split-joint represent a specific type of structure, and they are transformed into a node regrouping all services belonging to the parallel structure. The associated arcs are transformed into sequence arcs. The weight of the split arc is recalculated as:

Let $a_{i,j} : n_i \rightarrow n_j$ be an arc, if $a_{i,j}$ be a split type, then $w_{a_{i,j}} = (v_{obj_1}(f_{fl}), \dots, v_{obj_k}(f_{fl}))^T$. $a_{i,j} \leftarrow sequence$. If $a_{i,j} = splitJoin$, then $a_{i,j} \leftarrow sequence$, where f_{fl} is the

evaluation function of the flow structure. Finally, the MSCG transformation and the recalculation of arc weights allow building a suitable set of answering composition (called optimization search space), considering all the composition structures.

2) *Acyclic Composition Graph*: The acyclic composition graph (ACG) contains only sequences. Its nodes can be defined as nodes of MSCG, but to simplify the representation of node, one single type of node will be used and it contains variables for data URI (URI_{DA}), informative service URI (URI_{IS}), and active service URI (URI_{AS}).

$$n = \langle URI_{DA}, URI_{IS}, URI_{AS} \rangle \quad (6)$$

Where the service URIs, URI_{IS}, URI_{AS} , correspond respectively to the OWL-S description of informative and active services.

The arc weight values $w_{a_{j,j+1}}$, are calculated by considering all values of DA, IS and AS:

$$v_{obj_i}(n_{j+1}) = v_{obj_i}(DA) + v_{obj_i}(IS) + v_{obj_i}(AS).$$

The weight $w_{a_{j,j+1}}$ values of an arc $(a_{j,j+1} : n_j \rightarrow n_{j+1})$ are the sum of the destination node n_{j+1} elements (DA, IS, AS). These weights will be used in the optimization step, to evaluate the composition.

VI. COMPOSITION DESIGN ALGORITHMS

The goal of composition design algorithms is to build the composition graph answering the request, which will be used as an optimization search space. As explained in the previous section, when the composition design considers only sequence structures, the graph is acyclic (ACG); when all composition structures are taken into account, the graph is a MSCG. When the request functionalities dependency is global, the search space cannot be modeled as a graph. It is then modeled as a set of clusters. A cluster regroups the services matching a request functionality ($\mathcal{F}_{\mathcal{R}}[i]$), and a cluster is created for each request functionality. A request functionality can require an informative service (IS), active service (AS) or both. When an IS is used, it is executed. For each data, it generates a node is created. This node integrates the link (URI) of the informative service and the corresponding active service if it is required, and then it is affected to the corresponding cluster.

The proposed design composition algorithms, process progressively the request to build the executable composition graph. The process logic is based on the flooding algorithm and checks the composition structures. This checking is based on the characteristics of composition structures (as said in section III-C2). We present in the following two algorithms: algorithm 1 considers only sequence structures, and algorithm 2 considers all composition structure.

A. Composition Algorithms

We name the *current layer*, L_k , the set of nodes in the composition graph having a same depth level, being processed by the algorithm: $L_k = \{n_i\}$. Initially L_k contains the starting node sn of G , $L_k = \{sn\}$. One step of the algorithm corresponds to the full coverage of L_k . The node of L_k being processed during an iteration is named *current node* ($L_k[i]$), where i is the node index. We denote T the temporary buffer, L a layer of nodes, L_{end} the end layer, L_d a layer of data, d_i a condition and a_i a statement in a condition.

Algorithm 1: ACG: Design composition algorithm (Request \mathcal{R})

```

1:  $L_k, L_{k+1}, L_{End} \leftarrow \phi;$ 
2:  $L_k.add(sn(I_{F_{R_1}}));$ 
3:  $N.addAll(L_k); // Add all nodes of L_k;$ 
4:  $i \leftarrow 0;$ 
5: while  $i < |L_k|$  do
6:    $L_k[i] : current\ node;$ 
7:    $s_j \leftarrow SelectNextServices(L_k[i]); // Algorithm 3.$ 
8:   for  $s_j : NextService$  do
9:      $AddArcSequence(L_k[i], s_j);$ 
10:    if  $s_j \notin N$  then
11:      if  $EndFunctionality[s_j]$  then
12:         $L_{End}.add(s_j);$ 
13:      else
14:         $L_{k+2} \leftarrow RunInformativeService(s_j); //$ 
            $Algorithm\ 4.$ 
15:        for  $DA_k : L_{k+2}$  do
16:           $s_j \leftarrow Concatenation(s_j, DA_k);$ 
17:           $L_{k+1}.add(s_j);$ 
18:        end for
19:      end if
20:    else
21:       $L_{k+1}.add(s_j);$ 
22:    end if
23:  end for
24:   $N.addAll(L_{k+1}); L_k \leftarrow L_{k+1}; L_{k+1} \leftarrow \phi;$ 
25:   $i++;$ 
26: end while
27: Return ACG;

```

From a request, first fill in the current graph layer with matching services and process it (line 2 for starting and line 24 during processing). For each node in the current layer, select the next services according to their matching with functionalities F_{R_i} (Algo. 1 or Algo. 2 - line 7, call Algo. 3). The set of nodes created from next services is put into the next layer. When the current node output matches with one of the F_{R_i} outputs, the algorithm carries on with next node in the current layer. Otherwise, services having inputs matching the current node output are selected. Corresponding nodes are created after checking they do not already exist in the set of nodes N of G .

Algorithm 2: MSCG: Design Composition algorithm (Request \mathcal{R})

```

1:  $L_k, L_{k+1}, T, L_{End} \leftarrow \phi;$ 
2:  $L_k.add(sn(I_{F_{R_1}}));$ 
3:  $N.addAll(L_k);$ 
4:  $i \leftarrow 0;$ 
5: while  $i < |L_k|$  do
6:    $L_k[i]$  : current node;
7:    $s_j \leftarrow SelectNextServices(L_k[i]);$ 
8:   for  $s_j$  : NextService do
9:      $AddArcSequence(L_k[i], s_j);$ 
10:    if  $s_j \notin N$  then
11:      if  $EndFunctionality[s_j]$  then
12:         $L_{End}.add(s_j);$ 
13:      else
14:         $T.add(s_j);$ 
15:        if  $L_k[i] \in Split$  then
16:           $GetInSplit(s_j, L_k[i]);$ 
17:        end if
18:      end if
19:       $CheckSplitJoint(i, L_k, s_j);$ 
20:       $L_{k+2} \leftarrow RunInformativeService(s_j, D);$ 
21:      if  $|L_{k+2}| = 0$  then
22:         $SW \leftarrow CheckCondition(s_j, D);$ 
23:        if  $SW \neq null$  then
24:           $T.add(SW);$ 
25:        end if
26:      else
27:         $L_{k+1}.addAll(L_{k+2});$ 
28:      end if
29:    else
30:       $CheckSplitJoint(0, L_k, s_j);$ 
31:    end if
32:     $ChechSplit(L_k[i], T);$ 
33:     $L_{k+1}.addAll(T);$ 
34:     $T \leftarrow \phi;$ 
35:  end for
36:   $N.addAll(L_{k+1}); L_k \leftarrow L_{k+1}; L_{k+1} \leftarrow \phi; L_{k+1} \leftarrow \phi;$ 
37:   $i++;$ 
38: end while
39: Return MSCG;

```

An arc-sequence is created between the current node and each of the next nodes. When a selected service is an IS, it is invoked to obtain the data it provides before creating the arc (Algo. 1-line 14, Algo 2-line 20, call to Algo 5). Once the data are obtained (filling the data layer L_d), the algorithm creates an arc-sequence between the node of the service and each one of the data nodes. The selected service node is then replaced by the set of data nodes.

B. Sub-Branhes Used in Composition Algorithms

Algorithm 3: SelectNextServices (Service s_i)

```

1: if  $M(Output_{s_i}, Output_{F_{R_i}}) > \epsilon$  then
2:   if  $M(Input_{s_i}, Input_{s_j}) \leq \epsilon$  then
3:      $s_j$  selected;
4:     if  $M(Output_{F_{R_{i+1}}}, Output_{s_j}) \leq \epsilon \wedge |I_{\mathcal{R}}| = i + 1$ 
then
5:        $EndFunctionality[s_j] \leftarrow true;$ 
6:     end if
7:   end if
8: else
9:   if  $M(Input_{F_{R_{i+1}}}, Input_{s_j}) \leq \epsilon$  then
10:      $s_j$  selected;
11:     if  $M(Output_{F_{R_{i+1}}}, Output_{s_j}) \leq \epsilon \wedge |I_{\mathcal{R}}| = i + 1$ 
then
12:        $EndFunctionality[s_j] \leftarrow true;$ 
13:     end if
14:   end if
15: end if
16: Return selected sj;

```

When a next node has been newly created, i.e., the service corresponding to the node, is not selected already (Algo. 2, Algo. 5, Line 10), the algorithm checks the existence of a condition. This is the case if the output of the service represented by the node corresponds to one of the request conditions, or if the class of data represented by the node contains predicates used in a condition (Algo. 6). In this case, we create a SW-node and link it to the node by an arc-sequence. The node following the SW-node is then selected according to the first node output.

Algorithm 4: Run informative service (Service s)

```

1:  $L_d \leftarrow \phi;$ 
2: if  $s$ : informative Service then
3:    $L_{data} \leftarrow Runs(s);$ 
4: end if
5: for  $i = 1, |L_{data}|$  do
6:    $AddArcSequence(s, L_{data}[i]);$ 
7: end for
8: Return Ldata;

```

Algorithm 5: Run informative service (Service s , Condition D)

```

1:  $L_d \leftarrow \phi;$ 
2: if  $s$ : informative Service then
3:    $L_{data} \leftarrow Runs(s);$ 
4: end if
5:  $SW \leftarrow CheckConditionData(L_{data}[0], D);$ 
6:  $L_{data}.add(SW);$ 
7: for  $i = 1, |L_{data}|$  do
8:    $AddArcSequence(s, L_{data}[i]);$ 
9:   if  $SW \neq null$  then
10:      $AddArcSequence(L_{data}[i], SW);$ 

```

```

11:   end if
12: end for
13: Return Ldata;
Algorithm 6: CheckCondition (Node n, Condition D )
1: for i = 1, |L| do
2:   if (n is service) then
3:     if (a1 ∨ a2 ∨ ... ∨) ∈ class(Outputs) ∨ QoSs then
4:       SW ← CreateSwitchNode(di);
5:       AddArcSequence(L[i], SW);
6:     end if
7:   else
8:     if (a1 ∨ a2 ∨ ... ∨) ∈ DataClass(n) then
9:       SW ← CreateSwitchNode(di);
10:    end if
11:  end if
12:  T ← T ∪ SW;
13:  if (d1 ∨ d2 ∨ ... ∨ dn) ∈ class(Inputs) then
14:    AddArcSequence(SW, s);
15:  end if
16: end for
17: Return SW;

```

When all F_{r_i} have been covered, the next node is affected to the end layer (Algo. 2, Line 12). Otherwise, it is put into the next layer for (Algo. 1, Line 21), and for Algo. 2, it is put into temporary buffer, and later into the next layer (Line 33), after checking the split structure. The checking of split-structures is performed when the temporary buffer is full, containing all nodes matching the current node. The checking of split-joint-structure is performed when the next node is selected. Hence, the algorithm checks split-joint structure before the split structure. The process checks the existence of a split-joint structure starting from next node (Algo. 7). If it is selected from N of G , it is possible to find a node which can precede the next node. In this case, a complete check is performed (Algo. 2, Line 30), otherwise only a partial check is necessary (Algo. 2, Line 19). The complete check considers all nodes of the current layer. The partial check considers a current node and current layer nodes which have not been yet processed.

Algorithm 7: CheckSplitJoint (ServiceIndex i, NextLayer L_{k+1} , Service s)

```

1: IsSplitJoint ← false;
2: if M(Inputs, OutputL[i]) ≤ ε ∧ L[i] ∈ Split then
3:   for m = i + 1, |L| do
4:     if M(Inputs, OutputL[m]) ≤ ε ∧
      SameSplit(L[m], L[i]) then
5:       IsSplitJoint ← true;
6:       AddArcSplitJoint(L[i], L[m], s);
7:       GetOutSplit(L[m]);
8:     end if
9:     if (M(Inputs, OutputL[m]) > ε ∧ L[m] ∈ Split
then

```

```

10:      if SameSplit(L[m], L[i]) then
11:        GetInSplit(s, L[i]);
12:      end if
13:    end if
14:  end for
15:  if IsSplitJoint then
16:    GetOutSplit(L[i]);
17:  end if
18: end if

```

The algorithm 7 creates a split-joint-arc when it detects nodes, $L[i], L[m]$, having the same succeeding node (s : ends the parallel structure), and they have the same split-structure. After creation of split-joint-arc (Algo. 7, Line 6), the nodes leave the split structure (Algo. 7, line 7, line 16). The concatenation of parallel structures is possible, when paths of a same split-structure do not meet in a same split-joint structure. The node ending split s is then included in the parallel structure split (Algo. 7, line 11).

Algorithm 8: CheckSplit (Service s, Temporary buffer T)

```

1: for i = 1, |T| - 1 do
2:   for j = i + 1, |T| do
3:     if (M(OutT[i], OutT[j]) > ε) then
4:       AddArcSplit(s, T[i], T[j]);
5:       GetIntoSameSplit(T[i], T[j]);
6:     end if
7:   end for
8: end for

```

The checking for the existence of a split structure (Algo. 8) is performed between the current node and the nodes in the temporary buffer $T[j]$. If these nodes have different functionalities (i.e., different output), we create a split-arc and add them in the same split structure (Algo. 8, line 5). Then all following nodes are affected to this split structures (Alg. 2, line 15-16). When all nodes of the current layer are processed, the next layer becomes the current layer and so on until the next layer is empty. The algorithm terminates when this state is reached.

C. Algorithm Complexity

An algorithm complexity can be evaluated from two aspects. Time complexity measures the processing time, while Memory complexity measures the manipulated data size.

1) *Time Complexity*: In the algorithm 2, each request functionality ($\mathcal{F}_R[i]$) implies at least one iteration of the algorithm. This leads to a total of at least $card(\mathcal{F}_R)$ iterations. An iteration involves \bar{L} (\bar{L} is the average services layers size) discovered services, and for each discovered service there are $card(\mathcal{D})$ verifications of condition, $card(C_s)$ verifications of service constraint, and $O(split - joint)$ verifications of split-joint if the discovered service belongs to a split structure. For each iteration, the algorithm checks

$O(split)$ times the split structure. Discovered informative services are executed and all their provided data verified.

The algorithm has then a complexity order of $card(\mathcal{F}_R) \times \bar{L} \times card(\mathcal{D}) \times card(C_s) \times O(split) \times O(split-joint)$. Condition and constraint checking have a constant complexity, but the split

checking process has a complexity of *vector sort*, in the worst case $O(split) = \bar{L} \times log(\bar{L})$. The split-joint checking process has a complexity order \bar{L} . In conclusion, in the worst case, the algorithm complexity equals $O(card(\mathcal{F}_R) \times \bar{L} \times (\bar{L}^2 \times log(\bar{L})) \times card(\mathcal{D}) \times card(C_s))$. Since the discovered service \bar{L} , are important, then the complexity algorithm is cubic and has an order \bar{L}^3 .

The complexity of algorithm 1 equals $O(card(\mathcal{F}_R) \times card(C_s) \times \bar{L})$. This complexity is linear because $card(\mathcal{F}_R) \times card(C_s)$ is constant.

To conclude, our algorithms have a cubic complexity (Algo. 2) or linear complexity (Algo. 1). The most influential element is the number of discovered services. Therefore, the more services answering the request functionality, the more important is the resolution time. The number and severity of constraints put on services and data in the request will reduce the number of services. However too much or strict constraints can lead to empty layers in the composition graph.

2) *Memory Complexity*: It is important to reduce the memory used by the composition graph. Generally, we can store the graph nodes and their arcs, but we cannot save all paths of the graph, because the memory complexity is exponential (\bar{L}^l), where \bar{L} is the average layer size and $l = card(\mathcal{F}_R)$ is the layer number.

Concerning ACG, the nodes size is: $O(card(N)) = r + (l \times (2 \times \bar{L})) + 2 \approx l\bar{L}$, where $r = card(\mathcal{D})$ is the condition number. We multiply \bar{L} by 2, because each functionality can require both informative and active services, and (+2) corresponds to the start and the end node (*sn*, *en*). The arcs size is: $O(card(A)) = l \times (\bar{L}^2 + \bar{L} \times r) \approx l \times \bar{L}^2$,

Concerning MSCG, in the worst case, the nodes size is: $O(card(N)) = l \times r \times (\bar{L} + \bar{L}) + 2 = 2lr\bar{L}$. The arcs size is: $O(card(A)) = l \times (r \times \bar{L}^2) + r \times (\bar{L} + \bar{L}) = l \times r \times \bar{L}^2 + 2 \times l \times r \times \bar{L}$, which can be approximated by: $O(card(A)) \approx l \times r \times \bar{L}^2$. However, the memory complexity of a transformed MSCG is the same as for ACG.

VII. EXPERIMENTATION

The experiments we have conducted focus on three points: complex request description and services composition automatic design. For the request description, we give some example requests in natural language, and then describe manually their functionalities, conditions, constraints and objectives. For the automatic composition, we check its correctness by assessing the suitability of composition structures regarding the request and available services. The second point is the processing time of the composition design

algorithm. Based on its complexity, the size of matching services has an important impact for the processing time.

A. Implementation and Hypotheses

The used Web services are described semantically, using the OWL-S Language. For that purpose, we implemented a program generating OWL-S descriptions of services. This program relies on the Jdom API [42]. The type of service (input and output) is defined by referring to a domain ontology of travel booking. The values of QoS parameters are randomly generated: price and reputation of service are considered.

We used different APIs at the different levels of the resolution process. (1) Jena [43], and SPARQL [44] are used to check the data constraints (C_d) and conditions (\mathcal{D}). (2) The OWL-S API [45] is used to read the OWL-S service description and also to check services constraints (C_s). (3) Pellet [46] is used to check the matching level between *I/O* of services and the request, through the OWLS-MX API.

Our experiments have been conducted with the following set of hypotheses:

(1) Web services are formally described with OWL-S. We do not consider mapping with other Web services semantic description language.

(2) A QoS ontology is used within the OWL-S services descriptions, in order to describe the non-functional parameters such as price and reputation.

(3) A specific service domain ontology linked to a domain ontology is used to describe service I/O. In our experimentation cases, we have used the Travel Booking ontology which is associated to the MobileTTE Tourism ontology [47]. Request and services use this same ontology.

(4) We do not consider the necessary mapping of services described with different domain ontologies: all the services are described using a same travel booking ontology.

(5) As a consequence of (4), the matching process provides binary responses, even if the OWLS-MX API [12] that we use implements a full matchmaking process. Moreover only inputs and outputs parameters are exploited in the matchmaking.

Finally, the computing environment used to implement and test the application is a Core2 Duo CPU based machine (1.58 GHz) with 2.89 GB of RAM.

B. Complex Request Description

The tourism domain has been chosen as use case, because it offers a rich set of services. Moreover, the planning of a tourism trip is a classical example of service composition problem that anyone can face today. We consider the following request: “*We want to travel from City A to City B, to reserve several individual hotel rooms in destination city where each booking is billed separately, rent a car for six people and if a car does not allow six people, then rent two cars. I want to know the weather forecast and get a map*

of the destination city. Additionally, choose a museum visit or city monument visit according to the weather. The whole at best price and reputation, and the price must not exceed 3000 Euro".

This request has four functionalities: transport, hotel, car renting, city information. Each request functionality is characterized with the input and output class. These classes are detailed in Table III. Note that the city information functionality can be answered with different services such as, city weather, city map, and city museum service. These services can be executed in parallel because they have the same input type and provides different output types (as detailed in Table I).

This request contains three conditions, $\mathcal{D} = \{d_1, d_2, d_3\}$:

- (1) A condition referring to the hotel billing (d_1): "*each booking is billed separately*". This condition is written as, $d_1 : if (a_1 < a_2) \Rightarrow c_1 = false$, where a_1 is the reserved rooms number, a_2 is the billed reservations and c_1 is the achieved payment.
- (2) A condition referring to the weather state (d_2): "*choose the museum visit or city monument according to the weather*". This condition is written as, $d_2 : if (a_3 = good) \Rightarrow c_2 = c_3$. Where a_3 is the weather condition, c_2 is cultural activity and c_3 is monument visit.
- (3) A condition referring to the rental car type (d_3): "*when a car does not allow six people, then rent two cars*". This condition is written as, $d_3 : if (a_4 < 6) \Rightarrow c_4 = 2$. Where a_4 is the car capacity and c_4 is the number of cars.

The request predicates $a_1, a_2, a_3, a_4, c_1, c_2, c_3, c_4$, correspond to properties in the domain ontology. a_1 and a_2 are properties of the "BookedHotelInput" class. a_1 is the data property "numberOfrrooms", a_2 is the object property "paymentBookHotel". a_3 is a property of the "WeatherOutput" class, and corresponds to the data property "weather-condition". a_4 corresponds to the data property "carCapacity", and it is a property of the "CarInfo" class, and so one.

The request contains two objectives, price and reputation, and one constraint, the price. The price and reputation are described in the QoS ontology. Formally the request elements are: $I_{\mathcal{R}} = \{I_1, I_2, I_3, I_4\}$, $O_{\mathcal{R}} = \{O_1, O_2, O_3, O_4\}$, $\mathcal{D} = \{d_1, d_2, d_3\}$, $C = \{C_d, C_s, C_c\}$, $C_d = \emptyset$, $C_s = \emptyset$, $C_c = \{price\}$, $B = \{B_d, B_s, B_c\}$, $B_c = \{price, reputation\}$.

Table IV details some experimented requests. The initial request, described at the beginning of this section can be formalized with \mathcal{R}_2 or \mathcal{R}_4 , where the request functionalities order is different. The request \mathcal{R}_1 is less complex and corresponds to: "*I want to travel from City A to City B, reserve an hotel room in destination city and rent a car. The whole at best price and reputation, and the price does not exceed 3000 Euro*".

The request \mathcal{R}_3 corresponds to request \mathcal{R}_1 by adding the

requirements: "(1) reserve several hotel rooms and each booking is billed separately; (2) rent a car for six people and when a car does not allow six people, then rent two cars; (3) know the weather forecast and get a map for the destination city".

We choose these specific requests, in order to evaluate the correctness of the automatic composition, by analyzing the different compositions designed by the algorithm. The requests $\mathcal{R}_1, \dots, \mathcal{R}_4$ are described semantically with the OWL-CR ontology.

In the following section, we present our experiments and the obtained results. The experiments rely on different use cases, each use-case corresponding to a specific request.

C. Automatic Composition Results

Algorithm, 1 and 2 have been used to answer the request. In our tests, the matching threshold has fixed to 1, and the similarity threshold to 0.

The composition design algorithms search the available services in a UDDI, in order to design answer(s) for a given request. This UDDI contained references of services covering in particular the functionalities of the request: hotel, transport, rent a car and city information.

In order to evaluate the algorithm performances, we present and discuss two types of experiments. The first one concerns the correctness of composition. The second one concerns the composition performances in terms of response time.

1) Composition Correctness: The assessment of the composition correctness consists in verifying that the designed composition answers all request functionalities and detects correctly the composition structures whatever their numbers and the number of different ones.

Figures 5 and 6 illustrate a composition graph answering the request \mathcal{R}_1 , where the rectangle presents the *informative service*, a circle presents an *active service* (the same meaning for the following Figures), and the graph contains only the choice and sequence structures. The graph on Figure 5 contains 10 answering services by layer. For better visibility, the graph on Figure 6 contains only 2 answering composition by layers.

Observing the graph, it can be seen that, the transport request is answered either by an atomic service, *Booktransport*, or a composite service $\{Availabletransport \rightarrow BookFlight\}$. This illustrates that, for each request functionality, the algorithm can find one single matching service or builds sub-composition.

The Figure 7 illustrates the composition graph resulting from the processing of request \mathcal{R}_2 , which requires multiple composition structures: sequence, while, choice, switch, split and split-joint. The graph illustrates clearly the composition structures and for more visibility, each layer is limited to two services. We recall that arcs $\rightarrow, \vdash, \dashv$ illustrate respectively: sequence, split, and split-joint. The structure switch " sw_2 "

Class	Properties
$I_1 : InTransportClass$	$departCity, destinationCity, travelDate$
$I_2 : InHotelClass$	$roomNumber, arrivedDate, departDate$
$I_3 : InRentCarClass$	$dateTime4Take, dateTime4Make, InRentCarClass$
$I_4 : InCityInformation$	$cityName, Date$
$O_1 : OutTransportClass$	$flightOrTrainNumber, reservationNumber, placeNumber$
$O_2 : OutHotelClass$	$reservationNumber$
$O_3 : OutRentCarClass$	$reservationNumber$
$O_4 : OutCityInformation$	$Map, weather, Heritage, Museum$

Table III
INPUT AND OUTPUT CLASSES

Request	Formalization
\mathcal{R}_1	$< \{I_1, I_2, I_3\}, \{O_1, O_2, O_3\}, \mathcal{D} = \phi, \mathcal{C} = \{price\}, \mathcal{B} = \{price, reputation\}, \lambda = \phi >$
\mathcal{R}_2	$< \{I_1, I_2, I_3, I_4\}, \{O_1, O_2, O_3, O_4\}, \{d_1, d_2, d_3\}, \{price\}, \{price, reputation\}, \phi >$
\mathcal{R}_3	$< \{I_2, I_1, I_3, I_4\}, \{O_2, O_1, O_3, O_4\}, \{d_1, d_3\}, \{price\}, \{price, reputation\}, \phi >$
\mathcal{R}_4	$< \{I_2, I_1, I_4, I_3\}, \{O_2, O_1, O_4, O_3\}, \{d_1, d_2, d_3\}, \{price\}, \{price, reputation\}, \phi >$

Table IV
EXPEREMTED REQUEST

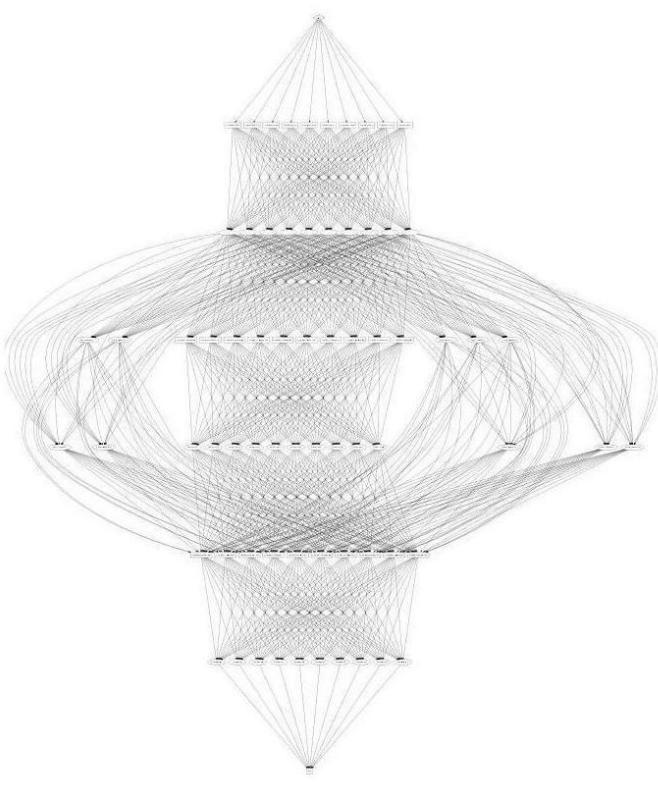


Figure 5. Executable service composition graph for \mathcal{R}_1 (10 services)

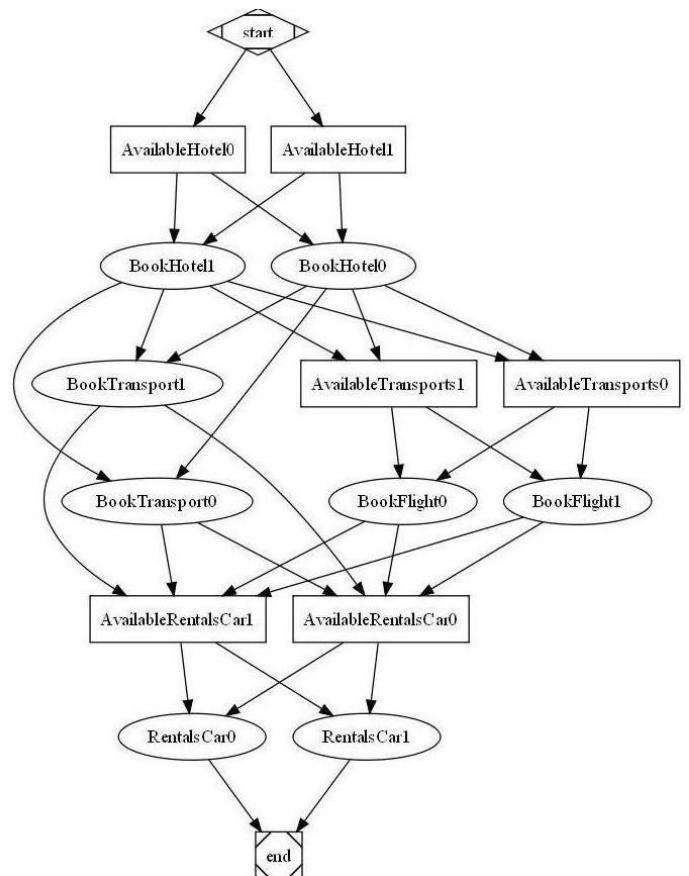


Figure 6. Executable service composition graph for \mathcal{R}_1 (2 services)

checks the city weather and it belongs to the parallel structure (between split and split-joint), this shows that the automatic composition algorithm can provide an imbrication of composition structures.

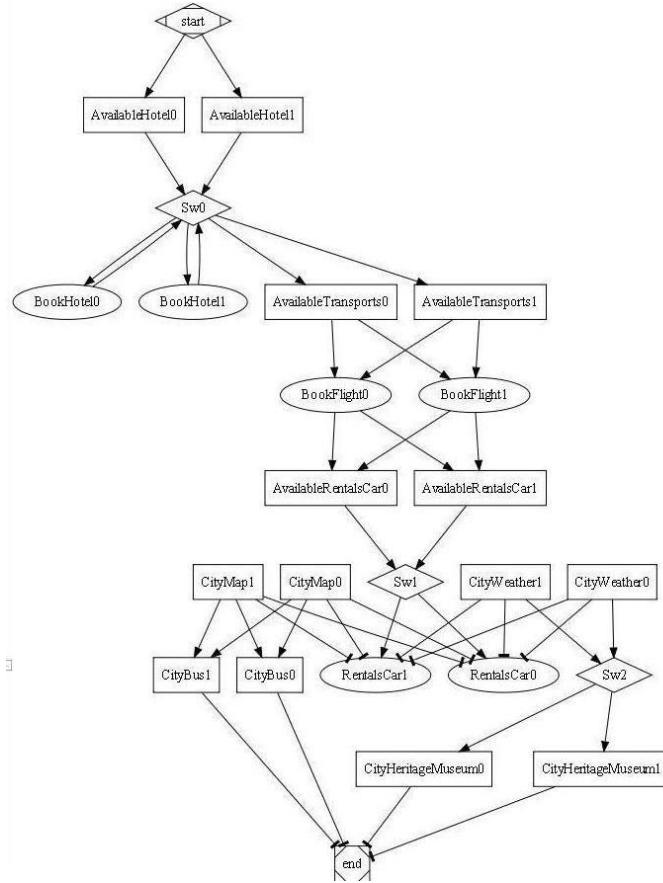


Figure 7. Executable service composition graphs for \mathcal{R}_2

The graph composition structures, illustrated Table V, are coherent with structure characteristics defined in Table I. For example, AF and AT are structured with a “choice” structure because their output classes match. AT and BF are structured with a “sequence” because the output of AT matches with the input of BF. BH is structured with a “while” because “each booking is billed separately”; the condition checking is before and after BH. A “switch” structure sw1 follows ARC service because the request condition “rent a car for six people” refers to the car capacity predicate and it belongs to the output of ARC. RC is followed by structure a “split”, opening parallelism, because the following services CW and CM have different functionalities. CB and CHM are followed by a “split-joint”, ending parallelism, because they are included in the same split structure and are followed by the same node “end”. The same holds for the other structures (Figure 8 and Figure 9).

In the built composition graphs, we have tested if the composition structures can be detected correctly. After this

Composition structure	participating services
Choice	(AH0,AH1), (BF0, BF1), (AH0, AH1), etc
Sequence	(AT0, BF0),(AT0, BF1),(BF0, AH0), etc
While	BH0, BH1
Switch	sw0, sw1, sw2
Split	RC0, RC1
Split-joint	end

Table V
STRUCTURES IN THE COMPUTED COMPOSITIONS,

where, AT denotes Available Transport, AF denotes Available Flight, BT denotes Book Train, BF denotes Book Flight, AH denotes Available Hotel, BH denotes Book Hotel, ARC denotes Available Rentals Car, RC denotes Rent Car , CB denotes City Bus, CW denotes City Weather, CM denotes City Map, and CHM denotes City Heritage Museum.

verification, we test the limit of this detection, such as, (1) test if the services following split structure can proceed, at the same time, the split-joint structure. (2) test if a parallel structure, split and split-joint can be in the middle of a composition path.

The Figure 8 and Figure 9 illustrate the composition graph resulting from the processing of request \mathcal{R}_3 , and \mathcal{R}_4 respectively. The view is limited to two services by layers for sake of readability.

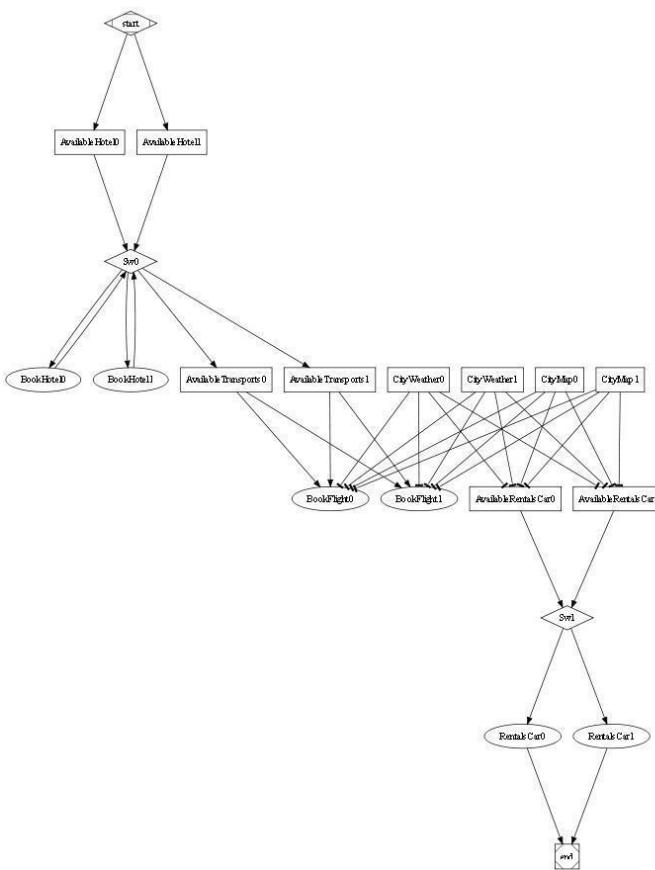
The graph on Figure 8 illustrates the services following a split and preceding a split-joint in the same time. In this test we have removed from the UDDI registry, the following services: ‘‘City Heritage Museum’’ and ‘‘City bus’’.

The graph on Figure 9 illustrates the possibility that the split and split-joint can be detected in the middle of the composition. The parallel structure follows BookFlight, and precedes AvailableRantelsCar.

2) *Performance Experiment:* As said in section VI-C1, the complexity of composition algorithms depend strongly on the answering services number (the width of its layers). We have conducted experiments to evaluate this influence.

Considering the composition graph answering the request having four request functionalities (\mathcal{R}_2), the Table VI summarizes the layer size impact on the composition design response time. Where $card(N)$, $card(A)$ are respectively nodes and arcs size of the composition graph, \bar{L} is the average layer size. We recall from the analysis made in section VI-C2, that $card(N) \approx 2lr\bar{L}$ and $card(A) \approx lr\bar{L}^2$, where r is the number of conditions of the request and l the number of layer. The Figure 10 illustrates the evolution of the execution time with respect to the average layer size.

If we consider that averagely each functionality can have about 15 answering services, the request processing requires 600 seconds (about 10 minutes) to return the MSCG and about 3 minutes to return ACG. As shown in Table VI and VII, the algorithm processing time depends on the number of answering services (\bar{L}) can be limited thanks to request constraints. We note that the larger the number of request constraints (services C_s and data C_d), the smaller

Figure 8. Executable service composition graph for \mathcal{R}_3

L	$card(N)$	$card(A)$	time (second)
2	24	42	25
4	45	124	57
6	65	246	89
8	85	408	108
10	105	610	194
12	125	852	333
14	145	1134	484
16	165	1456	740
18	185	1818	1098
20	205	2220	1641

Table VI

DISCOVERED SERVICES INFLUENCE ON RUNTIME WITH MSCG.

L	$card(N)$	$card(A)$	time (second)
2	16	32	28
4	30	120	49
6	44	264	73
8	58	464	94
10	72	720	119
12	86	1032	143
14	100	1400	171
16	114	1824	194
18	128	2304	220
20	142	2840	248

Table VII

DISCOVERED SERVICES INFLUENCE ON RUNTIME WITH ACG.

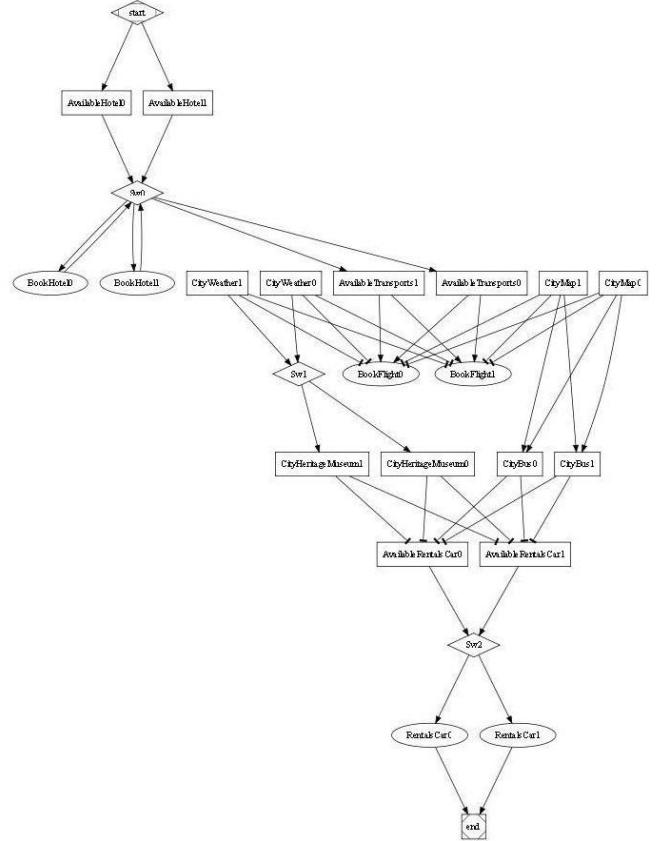
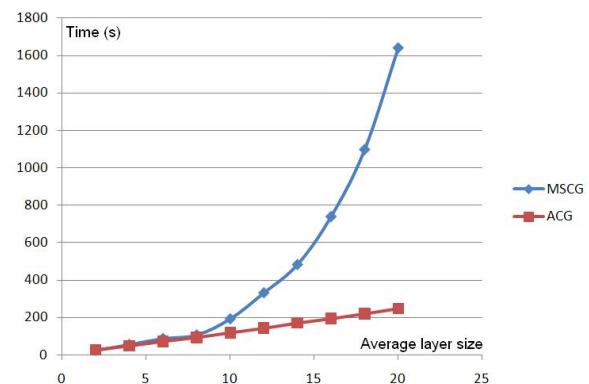
Figure 9. Executable service composition graph for \mathcal{R}_4 

Figure 10. Dependence between average layer size and execution time.

the number of answering services and then the smaller the processing time.

VIII. SPECIFIC CASES

The complex request resolution method changes according to the characteristics of composition graph, request dependencies type and dynamic events. In this section, we discuss these specific cases.

A. One Step and Multi Steps Resolution

The composition graph can be a layered digraph (order pair of sets, as illustrated in Figure 3) in which the nodes of two consecutive layers are all connected together, (i.e., there exists an arc $a_{u,v}$ between each node u and v of consecutive graph layers, l_k and l_{k+1}). This case is formalized by the following condition:

$$\forall i \in \{1 \dots k\}, \forall u \in l_i, \forall v \in l_{i+1} : a_{u,v} \in A. \quad (7)$$

At design time, the selection of compositions answering a request may or may not require an optimization method, depending on the composition graph structure and the number of objectives in the request. Considering the request functionalities have no global dependency, the composition does not require an optimization method when: (a) the graph layers respect the condition 7 and there is only a request objective ($\text{card}(\mathcal{B}) = 1$); or (b) the objectives can be combined because their importance levels λ are fixed ($\lambda_i = x, x \in [0, 1]$), thus allowing to order them. Optimization is not required when condition 7 is verified and the following condition is verified:

$$(\text{card}(\mathcal{B}) = 1) \vee (\text{card}(\mathcal{B}) > 1 \wedge \lambda \neq \phi \wedge \lambda_i = x) \quad (8)$$

In this case, the request functionalities can be treated separately, for each request functionality, and several matching services can be found. The more suitable one is selected according to the request objectives and constraints, and then executed. In other words, if this condition is verified, the steps: (2) service discovery, (3) composition design, (4) optimization of compositions, (5) execution of compositions, (illustrated in Figure 1) can be merged in one step.

In any other case, an optimization method is required to select the best composition. Therefore, we distinguish two types of request processing: one step resolution (no optimization) and multi-steps resolution (with optimization), as illustrated in figure 11.

The automatic composition is also adapted consequently: composition with selection for one step resolution, and composition without selection for multi-steps resolution. The compositions selection without optimization method means that this selection is done during the composition (left-hand side of Figure 12). When the composition selection is done with an optimization method, the composition design provides the set of composed services (right-hand side of Figure 12), in order to use it as an optimization search space.

The composition design steps are finally the following, the required steps depending on the need for optimization:

(0) Formalize user request with OWL-CR. (1) Read the request functionalities. Then for each functionality: (2) Find the corresponding services using the `discover` process,

which uses the matchmaker process for computing a matching level between request input/output and existing services. The discovered services and data are then filtered, according to the service constraints (c_s) and data constraints (c_d).

(3) The Composition design takes into account the possible composition structures by considering the request conditions \mathcal{D} .

(4) Select the best service according to the request objective(s) \mathcal{B} .

(4A) and (5A): Go to the next functionality, if the discovered (4A)/selected (5A) service fulfills the current functionality.

(4B) and (5B): consider again the same treated functionality, if the discovered (4B) or selected (5B) service does not fulfill the functionality. In this case, we consider the discovered or selected service output as a next request input.

B. Global Dependencies and Compositions Model

The existence of global dependencies generates some complexity in composition modeling. This is because the compositions graph will be designed with multiple layers of matching services and only subsequent layers can have relation arcs between them. Indeed, the checking at each node of the arcs history requires to maintain a list of all preceding paths to consider the relations with the following layers, which is a task having an exponential complexity [48]. The memory complexity for a node in the graph can be \bar{L}^l , where \bar{L} is the average number of layers and l is the number of layer. Let n_i, n_{i+1} be nodes; to check the existence of an arc $a_j : n_i \rightarrow n_{i+1}$, we may require to check conditions related to nodes $[n_1 \dots n_{i-1}]$.

We can neglect the path history by modeling the set of compositions as a graph, but without any guaranty regarding the validity of the composition. To be able to considering valid compositions only, we model the compositions as a set of clusters, each cluster corresponds to a set of services answering a request functionality. Then the validity of the compositions is checked during the initial step of the optimization.

C. Dynamic Events Processing

As said in section I-A, the steps (3-5: composition, optimization, execution) can be concerned by dynamical events, because on the one hand, random events can lead to service breakdown, affecting quality of service, and on the other hand because new services can appear. This impacts the composition design, the composition optimization and the composition execution.

During the composition design, perturbations can be neglected, because the composition design takes few time, and considers available services only.

During composition optimization, a specific mechanism has to be taken into account.

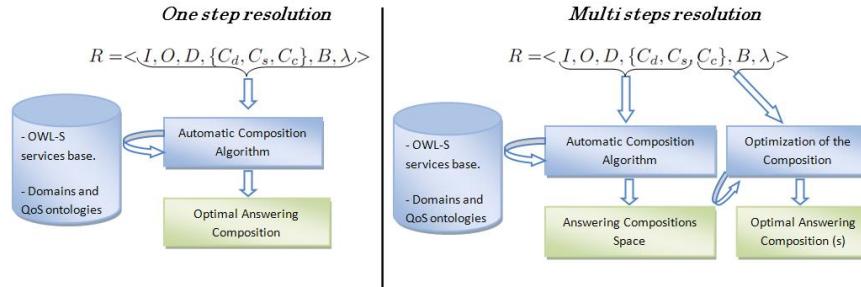


Figure 11. Solving request steps

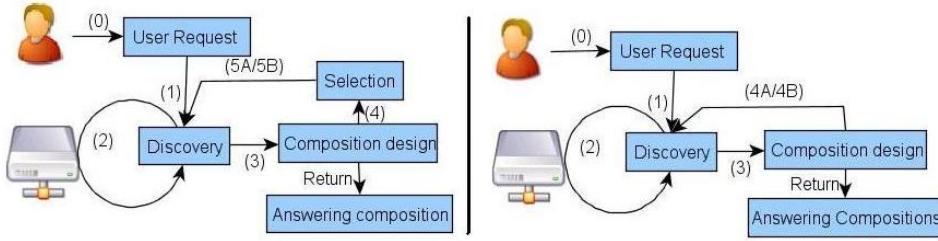


Figure 12. Automatic composition flow

During composition execution, the events are considered within the composition representation. When this one does not provide an alternative solution, we create a sub-composition corresponding to a new request and then select the best sub-composition with or without optimization. The new created request considers only the functionalities, that have not been fulfilled. We consider a functionality is completely achieved only when the corresponding active service is executed. Let i be an index of the last completed functionality, the new request is formalized as:

$$\mathcal{R} = \langle \mathcal{F}_{[i+1, \dots]}, \mathcal{D}, \mathcal{C}, \mathcal{B}, \lambda \rangle \quad (9)$$

Then we reuse an automatic composition algorithm (see section VI). In order to reduce the time of resolution, we suggest using algorithm 1 instead of algorithm 2, because it has lower resolution complexity.

To estimate the possibility of reparation, without recreating sub-composition that already exist in the graph, we calculate the k -connectivity of the composition graph.

The k -connectivity of a composition graph evaluates how many nodes can breakdown with assuring that the graph has the compositions alternatives. For example, if $k=3$, then for each depart-destination node, maximally two nodes can be deleted for assuring the possibility of finding an alternative path ($k - 1$: maximal number of nodes, whose can be deleted). For a fully connected composition graph, k corresponds to a minimal size of existing layers, $k = \min(\text{card}(L))$.

IX. CONCLUSION AND FUTURE WORK

This paper proposes a groundwork for solving complex Web requests with Web services composition considering the request processing steps. The set of compositions are modeled according to the request characteristics (request objectives and constraints). When there exist global dependencies, the set of possible compositions can not be modeled as a graph. In this case, compositions are modeled as a set of clusters. When the request dependency is not global, the set of compositions is modeled as a directed graph. The latter can be built with the algorithms we proposed.

Our algorithms answer a request, and in order to define answering compositions, exploit existing Web services and a semantic description of both request and services. If the request objectives or constraints have different evaluation function for different composition structures, then the composition algorithm 2 can be used. Otherwise, algorithm 1 is used. The composition graphs built by the algorithms, MSCG or ACG, can be used as a search space for the optimization process. In addition, if one of the composed services breaks down, the proposed algorithms support the repairing of the original composition.

We cited conditions allowing to choose automatically the suitable composition algorithm, suitable model of compositions and suitable resolution steps.

Experiments show the correctness of obtained compositions, concerning all composition structures, and the runtime performance. When only sequence structures are considered, the runtime complexity is linear with the average Web services. When all composition structures are considered,

the runtime complexity is cubic.

In the future work, we look: (1) to consider better the personalized knowledge of user, [49], [50] consider the composition answering a request as a personalized composition. According to our point of view, the answering composition is personalized when it answers a request by considering its conditions \mathcal{D} . The personalization can be widen, by considering the user profile for example. (2) To define the process allowing the transformation, from MSCA, ACG or clusters set to OWL-S or BPEL4WS description. This transformation does not consider the data (input values if services) and concerns only the non-personalized composition, in order to add them to the services base.

ACKNOWLEDGMENT

This work has been performed under the PhD grant TR-PhD-BFR07 funded by the Luxembourg National Research Fund (FNR).

REFERENCES

- [1] C. H. Marcussen, "Trends in European Internet Distribution - of Travel and Tourism Services," Tech. Rep., 2009. [Online]. Available: <http://www.crt.dk/uk/staff/chm/trends.htm>
- [2] B. Batouche, Y. Naudet, and F. Guinand, "Algorithm to solve web service complex request using automatic composition of semantic web service," in *COGNITIVE*, 2010, pp. 84 – 89.
- [3] J. Rao and X. Su, "A survey of automated web service composition methods," in *Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition, SWSWPC 2004*, 2004, pp. 43–54.
- [4] D. B. CLARO, "SPOC - Un canevas pour la composition automatique de services web ddis la ralisation de devis," Ph.D. dissertation, Universit d'angers, october 2006.
- [5] B. Jeong, H. Cho, and C. Lee, "On the functional quality of service (f qos) to discover and compose interoperable web services," *Expert Syst. Appl.*, vol. 36, pp. 5411–5418, April 2009. [Online]. Available: <http://portal.acm.org/citation.cfm?id=1497653.1498400>
- [6] Z. Li, L. O'Brien, J. Keung, and X. Xu, "Effort-oriented classification matrix of web service composition," in *Proceedings of the 2010 Fifth International Conference on Internet and Web Applications and Services*, ser. ICIW '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 357–362. [Online]. Available: <http://dx.doi.org/10.1109/ICIW.2010.59>
- [7] D. Martin, M. Burstein, D. Mcdermott, S. Mcilraith, M. Paolucci, K. Sycara, D. L. McGuinness, E. Sirin, and N. Srinivasan, "Bringing semantics to web services with owl-s," *World Wide Web*, vol. 10, pp. 243–277, September 2007. [Online]. Available: <http://dl.acm.org/citation.cfm?id=1285732.1285745>
- [8] C. REY, "D 2 CP et computeBCov: Un prototype et un algorithme pour la découverte de services web dans le contexte du web sémantique," *Ingénierie des systèmes d'information(2001)*, vol. 8, no. 4, pp. 83–112, 2003.
- [9] M. Klusch, B. Fries, and K. Sycara, "Automated semantic web service discovery with owl-s-mx," in *AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*. New York, NY, USA: ACM Press, 2006, pp. 915–922. [Online]. Available: <http://dx.doi.org/10.1145/1160633.1160796>
- [10] U. Rerrer-Brusch, "Service Matching with Contextualised Ontologies," Ph.D. dissertation, University of Paderborn, october 2006.
- [11] D. Martin, M. Burstein, E. Hobbs, O. Lassila, D. Mcdermott, S. Mcilraith, S. Narayanan, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, and K. Sycara, "OWL-S: Semantic Markup for Web Services," Nov. 2004. [Online]. Available: <http://www.w3.org/Submission/OWL-S/>
- [12] M. Klusch, B. Fries, and M. Khalid, "Owl-s-mx: Hybrid owl-s service matchmaking," in *Proceedings of 1st Intl. AAAI Fall Symposium on Agents and the Semantic Web*, 2005.
- [13] S. Narayanan and S. A. McIlraith, "Simulation, verification and automated composition of web services," in *WWW*, 2002, pp. 77–88.
- [14] R. Hull and J. Su, "Tools for composite web services: a short overview," *ACM SIGMOD Record*, vol. 34, no. 2, pp. 86–95, 2005.
- [15] A. Brogi and S. Corfini, "Behaviour-aware discovery of web service compositions," *International journal of Web services research*, Tech. Rep., 2006.
- [16] P. Xiong, Y. Fan, and M. Zhou, "Qos-aware web service configuration," *IEEE Transactions on Systems Man and Cybernetics Part A Systems and Humans*, vol. 38, no. 4, pp. 888–895, 2008.
- [17] X. Tang, C. Jiang, and M. Zhou, "Automatic web service composition based on horn clauses and petri nets," *Expert Syst. Appl.*, vol. 38, pp. 13 024–13 031, September 2011. [Online]. Available: <http://dx.doi.org/10.1016/j.eswa.2011.04.102>
- [18] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Q. Sheng, "Quality driven web services composition," *Proceedings of the 12th international conference on World Wide Web*, pp. 411–421, 2003.
- [19] X. X. Yifei Wang, Hongbing Wang, "Web Services Selection and Composition based on the Routing Algorithm," *10th IEEE International Enterprise Distributed Object Computing Conference Workshops*, pp. 57–66, 2006.
- [20] J. Cardoso and A. Sheth, "Semantic e-workflow composition," *Journal of Intelligent Information Systems*, vol. 21, pp. 191–225, 2003.
- [21] H. Levesque, F. Pirri, and R. Reiter, "Foundations for the situation calculus," pp. 159–178, 1998.
- [22] J. T. E. Timm, "Specifying semantic web service compositions using uml and ocl," in *In 5th International Conference on Web Services*. IEEE press, 2007.

- [23] R. Kazhamiakin and M. Pistore, "A parametric communication model for the verification of bpel4ws compositions," in *In Mario Bravetti, Lela Kloul, and Gianluigi Zavattaro, editors, EPEW/WS-FM, volume 3670 of Lecture Notes in Computer Science.* Springer, 2005, pp. 318–332.
- [24] D. PELLIER and H. Fiorino, "Un modle de composition automatique et distribue de services web par planification," in *Revue d'Intelligence Artificielle, volume 23, no. 1, 2009, pp. 13–46.*
- [25] S. R. Ponnenkanti and A. Fox, "Sword: A developer toolkit for web service composition," in *Proceedings of the 11th International WWW Conference (WWW2002), Honolulu, HI, USA, 2002.*
- [26] S.-C. Oh, B.-W. On, E. J. Larson, and D. Lee, "Bf*: Web services discovery and composition as graph search problem," in *e-Technology, e-Commerce, and e-Services, IEEE International Conference on, pp. 784–786, 2005.*
- [27] M. Klusch and A. Gerber, "Semantic web service composition planning with owls-xplan," in *In Proceedings of the 1st Int. AAAI Fall Symposium on Agents and the Semantic Web, 2005, pp. 55–62.*
- [28] E. M. Goncalves da Silva, L. Ferreira Pires, and M. J. van Sinderen, "An algorithm for automatic service composition," in *1st International Workshop on Architectures, Concepts and Technologies for Service Oriented Computing, ICSOFT 2007, Barcelona, Spain, E. M. Goncalves da Silva, L. Ferreira Pires, and M. J. van Sinderen, Eds. INSTICC Press, July 2007, pp. 65–74.*
- [29] S.-C. Oh, D. Lee, and S. R. T. Kumara, "Web service planner (wspr): An effective and scalable web service composition algorithm," *Int. J. Web Service Res., vol. 4, no. 1, pp. 1–22, 2007.*
- [30] C. Rey, "Dcouverte des meilleures couvertures d'un concept en utilisant une terminologie Application la dcouverte de services web smantiques," Ph.D. dissertation, Universit Blaise Pascal - Clermont II, dcembre 2004.
- [31] G. Alonso, F. Casati, H. Kuno, and V. Machiraju, "Web services. concepts, architectures and applications," 2003.
- [32] UDDI, "Uddi technical white paper," September 2000.
- [33] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, and D. Orchard, "Web services architecture," World Wide Web Consortium, Note NOTE-ws-arch-20040211, February 2004.
- [34] L. Bourgois, "Reprsentation et comparaison de Web services complexes avec des logiques dynamiques," Ph.D. dissertation, Universite Paris 13- Villetaneuse, june 2007.
- [35] J. Schaffner, H. Meyer, and M. Weske, "A formal model for mixed initiative service composition," *Services Computing, IEEE International Conference on, pp. 443–450, 2007.*
- [36] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana, *BPEL4WS, Business Process Execution Language for Web Services Version 1.1,* IBM, 2003. [Online]. Available: <http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel/ws-bpel.pdf>
- [37] S. V. Hashemian and F. Mavaddat, "Automatic composition of stateless components: a logical reasoning approach," in *Proceedings of the 2007 international conference on Fundamentals of software engineering, ser. FSEN'07.* Berlin, Heidelberg: Springer-Verlag, 2007, pp. 175–190. [Online]. Available: <http://portal.acm.org/citation.cfm?id=1775223.1775235>
- [38] L. H. P. G. Dan, A., "Web service differentiation with service level agreements," White Paper, IBM Corporation, Tech. Rep., March, 2003.
- [39] "<http://www.w3.org/Submission/WSDL-S/>," 12-12-2011.
- [40] R. Aggarwal, K. Verma, J. Miller, and W. Milnor, "Dynamic Web Service Composition in METEOR-S," 2004.
- [41] J. Cardoso, J. Miller, A. Sheth, and J. Arnold, "Modeling Quality of Service for Workflows and Web Service Processes," *Web Semantics Journal: Science, Services and Agents on the World Wide Web Journal, vol. 1, no. 3, pp. 281–308, 2004.*
- [42] "<http://www.jdom.org/docs/apidocs/>," 12-12-2011.
- [43] hp, "Jena - A Semantic Web Framework for Java," available: <http://jena.sourceforge.net/index.html>, 2002.
- [44] E. P. hommeaux and A. Seaborne, "Sparql query language for rdf (working draft)," W3C, Tech. Rep., March 2007. [Online]. Available: <http://www.w3.org/TR/2007/WD-rdf-sparql-query-20070326/>
- [45] "<http://www.mindswap.org/2004/owl-s/api/>," 12-12-2011.
- [46] E. Sirin, B. Parsia, B. Grau, A. Kalyanpur, and Y. Katz, "Pellet: A practical OWL-DL reasoner," *Web Semantics: Science, Services and Agents on the World Wide Web, vol. 5, no. 2, pp. 51–53, Jun. 2007.* [Online]. Available: <http://dx.doi.org/10.1016/j.websem.2007.03.004>
- [47] J.-D. Labails, "Description de l'ontologie des ressources touristiques," MobileTTE WP5.1: Standard d'interoprbilit pour les donnes touristiques, Technical Report, Henri Tudor Public Research Center, Luxembourg, Tech. Rep., 2006.
- [48] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* New York, NY, USA: W. H. Freeman & Co., 1979.
- [49] P. Albers and O. Licchelli, "Composition de services web personalis," in *Intelligence Artificielle et Web Intelligence Conference, Grenoble, July 2007.*
- [50] S. Khapre and D. Chandramohan, "Personalized web service selection," in *International Journal of Web Semantic Technology (IJWesT) Vol.2, No.2, April 2011.*

Scripting Technology for Generative Modeling

Christoph Schinko[¶], Martin Strobl[¶], Torsten Ullrich[§], and Dieter W. Fellner^{¶‡}

[¶] Institut für ComputerGraphik & WissensVisualisierung, Technische Universität Graz, Austria

c.schinko@cgv.tugraz.at, m.strobl@cgv.tugraz.at

[§] Fraunhofer Austria Research GmbH, Graz, Austria

torsten.ullrich@fraunhofer.at

[‡] GRIS, TU Darmstadt & Fraunhofer IGD, Darmstadt, Germany

d.fellner@igd.fraunhofer.de

Abstract—In the context of computer graphics, a generative model is the description of a three-dimensional shape: Each class of objects is represented by one algorithm M . Furthermore, each described object is a set of high-level parameters x , which reproduces the object, if an interpreter evaluates $M(x)$. This procedural knowledge differs from other kinds of knowledge, such as declarative knowledge, in a significant way. Generative models are designed by programming. In order to make generative modeling accessible to non-computer scientists, we created a generative modeling framework based on the easy-to-use scripting language *JavaScript* (JS). Furthermore, we did not implement yet another interpreter, but a JS-translator and compiler. As a consequence, our framework can translate generative models from *JavaScript* to various platforms. In this paper we present an overview of *Euclides* and quintessential examples of supported platforms: Java, Differential Java, and GML. Java is a target language, because all frontend and framework components are written in Java making it easier to be embedded in an integrated development environment. The Differential Java backend can compute derivatives of functions, which is a necessary task in many applications of scientific computing, e.g., validating reconstruction and fitting results of laser scanned surfaces. The postfix notation of GML is very similar to that of Adobes Postscript. It allows the creation of high-level shape operators from low-level shape operators. The GML serves as a platform for a number of applications because it is extensible and comes with an integrated visualization engine. This innovative meta-modeler concept allows a user to export generative models to other platforms without losing its main feature – the procedural paradigm. In contrast to other modelers, the source code does not need to be interpreted or unfolded, it is translated. Therefore, it can still be a very compact representation of a complex model.

Keywords-Generative modeling, procedural modeling, computer graphics, *JavaScript*, compiler

I. INTRODUCTION

Offering an easy access to programming languages that are difficult to approach directly reduces the inhibition threshold dramatically. Especially in non-computer science contexts, easy-to-use scripting languages have gained a lot of attention in the past few years.

In the context of Cultural Heritage, the Generative-Modeling-Language (GML) is an established procedural modeling environment designed for expert users. The aim

of the *Euclides* modeling framework is to offer an easy-to-use approach to facilitate these platforms. The translation mechanism for GML within *Euclides* has already been described in “Euclides – A JavaScript to PostScript Translator” and presented at the International Conference on Computational Logics, Algebras, Programming, Tools, and Benchmarking [1].

Originally, scripting languages like *JavaScript* were designed for a special purpose, e.g., to be used for client-side scripting in a web browser. Nowadays, the applications of scripting languages are manifold. *JavaScript*, for example, is used to animate 2D and 3D graphics in VRML [2] and X3D [3] files. It checks user forms in PDF files [4], controls game engines [5], configures applications, and performs many more tasks. According to J. K. OUSTERHOUT scripting languages use a higher level of abstraction compared to system programming languages as they are often typeless and interpreted to emphasize the rapid application development purpose [6]. Whereas system programming languages are designed for creating algorithms and data structures based on low-level data types and memory operations. As a consequence, graphics libraries [7], graphics shaders [8] and scene graph systems [9], [10] are usually written in C/C++ dialects [11], and procedural modeling frameworks use scripting languages such as *Lua*, *JavaScript*, etc.

A. Geometric Modeling

When describing the shape of three-dimensional objects, two different approaches are established:

- composing an object of basic primitives (points, triangles, quads, etc.),
- creating a procedural description [12].

A composition of primitives can be achieved by conventional geometric modeling or by using 3D acquisition devices, which are always more or less noisy. Whereas, a procedural description is based on an ideal object rather than a real one and is often used to describe an object's inherent properties. Its strength lies in a very compact description, which, compared to conventional geometric descriptions, is not dependent on the number of primitives but on the

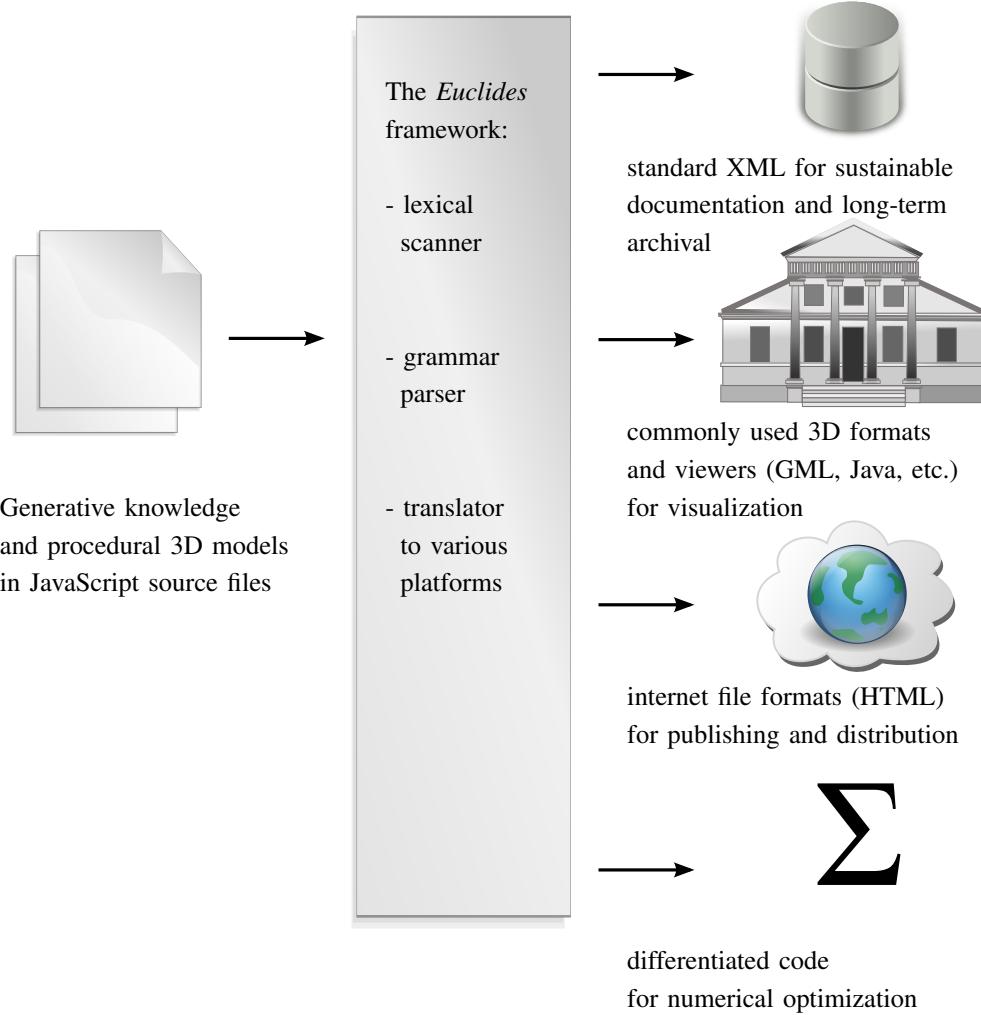


Figure 1. The meta-modeler approach of the *Euclides* framework has many advantages. In contrast to script-based interpreters, *Euclides* parses and analyzes the input source files, builds up an abstract syntax tree (AST), and translates it to the desired platform. Its platform and target independence as well as various exporters for different purposes are the main characteristics of *Euclides*.

model's complexity itself. However, generative models must not be seen as a replacement for established geometric descriptions, but as a semantic enrichment. Another advantage of procedural modeling techniques is the included expert knowledge within an object description. For example, classification schemes used in architecture, archaeology, civil engineering, etc. can be mapped to procedures, thus making the object easily identifiable by digital library services (indexing, markup and retrieval). For a specific object only its type and its instantiation parameters have to be identified. In combination, these two methods can be used to perform detailed mesh comparisons, which can reveal the smallest changes and damage of digitized artifacts. Such analysis and documentation tasks are valuable in the context of cultural heritage [13], [14].

B. Cultural Heritage

Procedural modeling techniques are well studied in the fields of computer-aided design and engineering. Unfortunately, in the context of cultural heritage, model complexity, model size, and imperfection have dimensions several orders of magnitude higher than many other fields of applications [15]. Cultural heritage artifacts often have a high inherent complexity, since they represent masterpieces of the human creative genius. As such artifacts are seldom single findings, they are therefore often embedded in larger excavation sites. An archaeological excavation may have an extent on the scale of kilometers with a high richness of detail on the scale of millimeters. Domain knowledge by cultural heritage experts and procedural modeling techniques are keys to cope with this complexity and size [16].

Generative modeling inherits methodologies of 3D modeling and programming, which leads to drawbacks in usability and productivity. The need to learn and use a programming language is a significant inhibition threshold especially for archaeologists, cultural heritage experts, etc. who are seldom experts in computer science and programming. The choice of the scripting language has a huge influence on how easy it is to get along with procedural modeling. This is why we use JavaScript – a beginner friendly, structured language.

C. JavaScript

JavaScript features a rather intuitive syntax, which is easy to read and to understand. A comprehensible, well-arranged syntax is useful, since source code is more often read than written. JavaScript supports features like dynamic typing and first-class functions. The most important feature is, that it is wide-spread amongst non-computer scientists – namely designers and creative coders. This is the reason why there are numerous tutorials available on the internet, resulting in an easy access to the language. JavaScript is used in many different environments and has evolved from being used only in a web-browser to a flexible multi-purpose scripting language. Our integrated scripting solution adds another chapter to the history of JavaScript usage.

Our meta-modeler approach *Euclides* is based on JavaScript and differs from other modeling environments in a very important aspect: target independence. Usually, a generative modeling environment consists of a script interpreter and a 3D rendering engine. A generative model (3D data structures with functionality) is interpreted directly to generate geometry, which is afterwards visualized by an integrated rendering engine.

In our system a model's source code is not interpreted but parsed into an intermediate representation, an abstract syntax tree (AST). After a validation process it is translated into a target language. The process of

parsing → validating → translating

offers many advantages as illustrated in Figure 1. The validation step involves syntax and consistency checks. These checks are performed to ensure the generation of a correct intermediate representation and to provide meaningful error messages as early as possible within the processing pipeline. The translation step, like every compilation/translation (see Section II, Related Work), consists of a parser frontend (see Section III, JavaScript Frontend), middleware, and backend (see Section IV, Target Backends), and offers platform independence (see Section V, Conclusion and Future Work). The same code basis can be translated into different languages for various purposes.

II. RELATED WORK

A. Generative Modeling

Procedural modeling systems often rely on grammars to describe the rules behind generative components. Early systems based on grammars were Lindenmayer systems [17], or L-systems for short. These systems provide the means for modeling plants, where they were successfully applied. Starting with simple strings, complex strings are created by using a set of string rewriting rules. A predefined set of rules is applied to an initial string forming a new, possibly larger string. The L-systems approach reflects a biological motivation. In order to use L-systems to model geometry an interpretation of the generated strings is necessary. The modeling power of these early geometric interpretations of L-systems was limited to creating fractals and plant-like branching structures. This leads to the introduction of parametric L-systems. The idea is to associate numerical parameters with L-system symbols to address continuous phenomena, which were not covered satisfactorily by L-systems alone.

CGA Shape, CityEngine: L-systems in combination with shape grammars are successfully used in procedural modeling of cities [18]. Parish and Müller presented a system that generates a street map including geometry for buildings given a number of image maps as input. For that purpose L-systems have been extended to allow the definition of global objectives as well as local constraints. However, the use of procedurally generated textures to represent facades of buildings often results in a limited level of detail. In later work, Müller et al. describe a system [19] to create more detailed facades based on a split grammar called *CGA shape*. A framework called the *CityEngine* provides a modeling environment for *CGA shape*. It relies on different views to guide an iterative modeling process.

Lipp et al. presented another modeling approach [20] following the notation of Müller [21] that deals with the aspects of more direct local control of the underlying grammar by introducing visual editing. The idea is to modify elements selected directly in a 3D-view, rather than editing rules in a text based environment. Principles of semantic and geometric selection are therefore combined as well as functionality to store local changes persistently over global modifications.

Model Graphs: A modeling method as well as a graphical user interface for the creation of natural branching structures was proposed by Lintermann et al. [22]. The idea is to represent the modeling process with a structure tree, which can be altered using specialized components describing geometry as well as structure. Another set of components can be used for defining global and partial constraints. These components are described procedurally using creation rules, which include recursion. Geometric data is generated according to the structure tree via a tree

traversal, where the components generate their geometrical output themselves.

Ganster et al. propose a procedural modeling approach [23] based on structure trees as well. They describe an integrated framework relying on a visual language. The infix notation of the language requires the use of variables, which are stored on a heap. A graph structure represents the rules used to create an object. Special nodes allow the creation of geometry, the application of operators as well as the usage of control structures. Various attributes can be set for nodes used in a graph. Directed edges between nodes define the order of execution, in contrast to a visual data flow pipeline where data is transported between the different stages.

Hierarchical Description: Finkenzeller presented another approach for detailed building facades [24] called *ProcMod*. It features a hierarchical description for an entire building. In order to create a building, the user provides a coarse outline and a basic style of the building including distinguished parts. The system then generates a graph representing the building. In the next step, the graph is traversed and geometry for every element of the graph is generated. This results in a detailed scene graph, in which each element can be modified afterwards. The version described has some limitations: for example, organic structures and inclined walls cannot be modeled.

Postfix Expressions: Havemann proposes a stack based language called *Generative Modeling Language* (GML), which allows, but is not limited to, creating polygonal meshes [25]. The postfix notation of the language is very similar to that of *Adobe Postscript*. High-level shape operators are created from low-level shape functionality. The GML serves as a platform for a number of applications, because it is extensible and comes with an integrated visualization engine.

An extended system presented by Mendez et al. combines semantic scene-graph markups with generative modeling [26]. The purpose of the system is the generation of semantic three dimensional models of underground infrastructure. A geospatial database and a rendering engine are combined in order to create an interactive application. The GML is used for on-the-fly generation of procedural models in combination with a conventional scene graph system with semantic markup.

Scripted Modelers: In contrast to specialized generative modelers, there are a number of 3D modeling software packages available like Autodesk MayaTM or 3ds MaxTM. They provide a variety of tools for modeling with polygons, non-uniform rational B-splines (NURBS) and predefined primitives. In addition to a graphical user interface (GUI), a scripting language is supplied to extend the functionality. It enables tasks that cannot be achieved easily using the GUI and speeds up complicated or repetitive tasks.

Processing and Grasshopper: Processing stands for a programming language and a development environment. It

was initially created to serve as a software sketchbook and to teach students fundamentals of computer programming [27]. It quickly developed into a tool that is used for creating visual arts. Processing is basically a Java-like interpreter offering a new graphics and utility API together with some usability simplifications. A large community behind the tool produced over seventy libraries to facilitate computer vision, data visualization, music, networking, and electronics.

Another tool with a creative background is Grasshopper [28]. Its main purpose is the creation of graphical algorithms. It is a graphical editor for Rhino's 3D modeling tools designed to be used without programming skills - unlike RhinoScript. In Grasshopper programs are created by dragging components onto a canvas and interconnecting these components. Many components create, but are not limited to, 3D geometry. Similar to Processing, there is a large community behind the tool.

B. JavaScript

JavaScript started as a simple client-side scripting language. Nowadays, there are a number of projects that use JavaScript in innovative ways. EMSScripten (<https://github.com/kripken/emscripten>) is a LLVM to JavaScript compiler. LLVM stands for low level virtual machine, and is an intermediary representation for code compiled from languages such as C, C++ or Objective-C. LLVM output used by EMSScripten is similar to assembly language.

The difficulty in translating an LLVM AST to JavaScript is that the high level representation of the source languages is lost. A translator hence needs to compact ambiguous assembly statements into general, high level language code. The EMSScripten algorithm, called relooper, produces output that models a virtual machine. The heap of the virtual machine is a huge array. A growing machine's stack is modeled as a variable that serves as an index to the heap. Necessary control flow such as *goto* and arbitrary labels are modeled as looping switch-statements. EMSScripten author Zakai is demonstrating the versatility of the transpiler by porting the computer game DOOM to JavaScript. The JavaScript output of compiled DOOM is accelerated by the optimizing compiler *GClosure* by Google.

Similar to a virtual machine, Bellard has written a PC emulator that runs in JavaScript. It uses JavaScript's typed arrays to emulate a feature-stripped 486 CPU. Typed arrays allow to apply views on an existing array, by which a programmer can access the array contents as differently sized chunks. This way, an array of 32 bit sized numbers might be accessed as 16 bit array. This way of addressing allows extracting single bytes from an array of integers. Obviously, bit-level handling is possible and facilitated, if one uses a byte to represent a machine-bit, scaling available memory by a factor of 8. To demonstrate the power of the emulator, Bellard shows a version of a small GNU/Linux

distribution that is properly executed by his JavaScript-based PC emulator.

III. JAVASCRIPT FRONTEND

A. Lexer and Parser

Despite the name, JavaScript is unrelated to the programming language Java, even if it copies many names and naming conventions. It is a functional programming language with support of structured programming constructs in C-style (e.g., if-statements, for-loops, switch-statements). In analogy to C, JavaScript differentiates between expressions and statements. However, there are a few important aspects that need to be mentioned:

- In contrast to C-style block-level scoping, JavaScript supports function-level scoping.
- Types are dynamic and they are associated with values; i.e., a variable's value defines its type.
- Functions are objects themselves and therefore can be assigned to variables, returned by functions, passed as arguments, and manipulated like any other object [29].

As JavaScript is typically interpreted, its design reflects interactivity. It relies on a run-time environment, i.e., an object model provides the functionality to communicate with the host environment. Furthermore, the default entry point for a parser is a *statement* rule as visualized in Figure 2. It does not have a mandatory, enclosing class structure or a main function as entry point. Additionally, the interactivity is reflected in required forward declarations, which can be created via forward references, as functions are first-class citizens.

The *statement* rule is split up in several sub-rules like *statementIf*, *statementDoWhile*, *statementExpression*. While these rules are rather straightforward, the *statementNativeCode* rule is a special feature of our grammar. It allows to embed native code into JavaScript. When JavaScript code gets translated, it is sometimes necessary to embed code written in the target language – so called native code. As a consequence, each platform-dependent library is available in all target versions. An illustrating example is taken from the Java version of the *Euclides' IO* library. It shows how writing output to the text console is handled.

The function `io_stdout_write` writes a text message to Standard Out. If necessary the parameter `msg` will be converted to a string. In order to be JavaScript compliant, native code is embedded in comments using the special character sequence `/*% */` to denote the beginning of a native code section. A similar mechanism is used to allow parsing annotations, where the special character sequence `/*@ */` is used to denote an annotations section inside a comment. Annotations are used similar to preprocessor directives in C. They are evaluated by the parser and additionally embedded in the target source code.

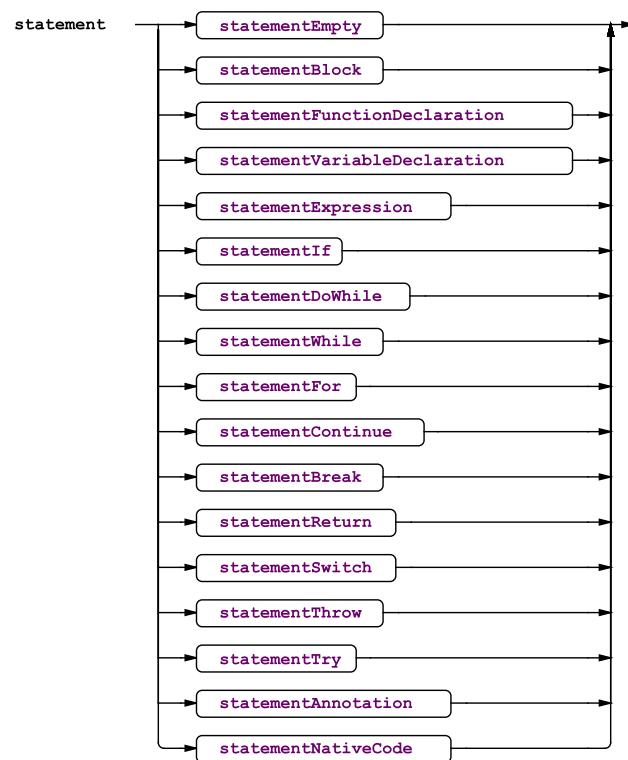


Figure 2. The entry point for a JavaScript parser is a *statement* rule, because JavaScript is typically interpreted in a run-time environment – statement by statement. As a consequence, the JavaScript grammar does not contain any enclosing class structures.

```

/*
  euclides.suppress_warning_unreferenced_o ←
object.io_stdout_write;
*/

/**
  This function writes a text message to
  standard out. The parameter 'msg' will be
  converted to a 'string', if necessary.
*/
function io_stdout_write(msg)
{
  /**
  System.out.print(usr_msg.toString());
  */
}
  
```

In the example above, the annotations technique is used to suppress a warning of an unreferenced object, which might occur, if the function is not used.

B. Abstract Syntax Tree

Our parser for JavaScript is written using ANother Tool for Language Recognition (ANTLR). ANTLR provides a framework for constructing recognizers, interpreters, compilers, and translators from grammatical descriptions [30]. It introduces a strategy called *LL(*)* parsing, which extends the *LL(k)* parsing strategy with lookahead of arbitrary length without explicitly specifying it. The purpose of using this framework is to syntactically and semantically check the provided input for JavaScript compliance and at the same time to generate an intermediate representation: an abstract syntax tree.

The AST offers three entry points into a script. The first entry point is the “obvious” representation: a sequence of statements. Each statement contains all included substatements and expressions as well as associated comments. The leaves of the AST store tokens together with formatting information (line and position). This tree structure is extended by reference and occurrence links; e.g., each method call references the method definition and each variable definition links to all its occurrences.

The list of all variables represents a second entry point to explore the AST. Each entry consists of the variable name, the statement in which it is defined, the scope of the variable as well as all occurrences in the source code.

Last but not least, the third entry point lists all function implementations including anonymous functions with complete function body. Similar to the contents of the variable list, each entry offers the function name, its defining statement, the scope of the function, the number of parameters this function takes, as well as all occurrences in the source code.

These structures are not only needed during the translation process, but they are valuable inspection tools. *Euclides*’ automatic documentation system exports these views and data structures in a collection of XHTML files: Using markup techniques directly jumping between occurrences and definitions of variables, function, etc. is possible; e.g., a screenshot of the automatic documentation created for the fibonacci example

```
function fibonacci(index) {
    switch (index) {
        case 0:
        case 1:
            return 1;
        default:
            return fibonacci(index-2)
                + fibonacci(index-1);
    }
}

var fibs = fibonacci(42);
```

The screenshot shows the Euclides documentation interface for the fibonacci example. The Variables tab is selected. It displays a table with two columns: Globally Defined and Locally Defined. Under Globally Defined, there is one entry for 'fibonacci'. Under Locally Defined, there are four entries, each corresponding to a line of code in the source file. The properties listed for each entry include explanation, declaration statement, unique name, name including scope, and references. The references column shows the line numbers where the variable is used throughout the code.

Figure 3. The *Euclides* documentation target represents JavaScript as a sustainable, standard-conform XML document can be displayed in an arbitrary web browser.

is shown in Figure 3. All globally and locally defined variables are listed in the Variables view. Several properties are available for each variable:

- Comments. Any comments associated with a variable are preserved and included.
- Location. The line of code (source, its line number and file name) where the variable is declared.
- Visibility. The name together with the scope, in which the variable is available.
- References. All references and uses of the variable in the source code including file name, line number and declaration statement.

Similarly to the Variables view, the Functions view is a collection of all functions defined in the source code and consists of the same four properties mentioned above. The Statements view is a collection of all statements of the source code together with filename and line number, which allows, for example, identifying duplicate code snippets. Also it gives a nice overview of the complexity of the source code. In the files view, the source code is available as XML document.

C. Libraries

A collection of libraries for geometry (*GEO*), graphical user interfaces (*GUI*), input/output (*IO*), mathematics (*MATH*) and some utilities (*UTL*) are available for the *Euclides* framework. They offer functionality to conveniently create generative models together with basic user interface elements. A simple example visualization of a Sierpinski

tetrahedron with GUI components to control the subdivision depth is shown in Figure 4.

IV. TARGET BACKENDS

When translating source code into target language code [31], the need to establish a proper naming standard quickly arises. A runtime environment is implemented using symbols and constructs available in the target language. Naming these constructs may interfere with the naming of code to be translated. For example, there may very likely be a function called `main` in the runtime environment rendering this name to be reserved. In order to overcome these limitations, all names in the translated code are extended with the prefix `usr_`, if it corresponds one-to-one to a name in JavaScript. Otherwise it is prefixed with `sys_`.

Additionally, dealing with different target languages not only means dealing with naming issues, but also dealing with character encoding. A character may be allowed to be used in JavaScript, but forbidden in a target language. As a consequence, we introduced lists of allowed characters and rules for character replacements to handle all naming issues.

These two mechanisms ensure the validity and consistency concerning naming and encoding issues of the generated code. The result of a name translation is always a name, which is

- 1) valid in the target language and
- 2) does not collide with any predefined names, name spaces, or keywords.

A. Java

Although Java and JavaScript have some similarities, the concepts of both languages show major differences. Java is a statically typed, class-based, general-purpose programming

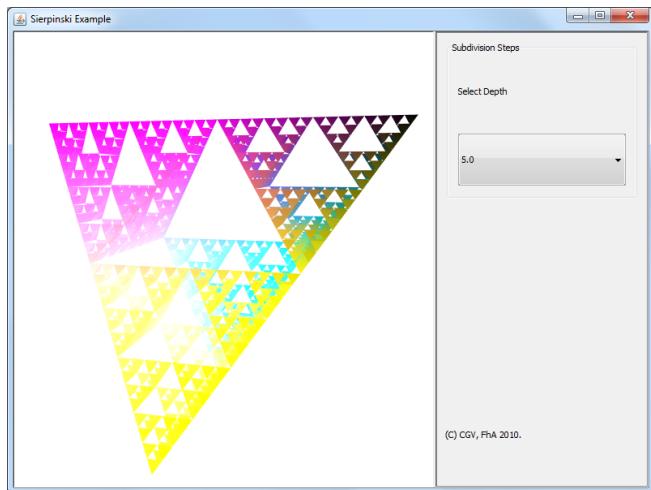


Figure 4. This figure shows how a Sierpinski tetrahedron example translated to the Java target looks like. A small GUI with a drop-down list to select the subdivision depth together with a 3D view of the Sierpinski tetrahedron at subdivision depth five is scripted using *Euclides*.

language designed to have a minimum of implementation dependencies to be able to follow the credo: “write once, run anywhere”.

It is chosen as a target language, because all frontend and framework components are written in Java making it easier to be embedded in an integrated development environment.

Data Types: Because of conceptual differences in the typing system, it is unpractical to project JavaScript data types onto built-in Java data types. For example, JavaScript makes no difference between integer numbers or floating-point numbers. There is just one data type called `number` that may hold any type of number. Similar difference can be found when comparing the remaining data types.

Dynamic typing is another big difference between the two languages. As a consequence, each JavaScript data type is re-built in Java to match its functionality making a total of seven data types. These data types are wrapped in a class called `Var`, which provides the properties,

- `getType()`
- `length(int ii)`

access functions

- `accessArray(int ii, Var index)`
- `accessObject(int ii, String attribute)`
- `assign(int ii, Var variable)`
- `delete(int ii, Var variable)`
- `executeDirect(int ii, Var THIS, Var[] parameters)`
- `executeIndirect(int ii, String attribute, Var[] parameters)`

and conversion methods applicable to all JavaScript variables:

- `toArray()`
- `toBoolean()`
- `toFunction()`
- `toNumber()`
- `toObject()`
- `toString()`
- `toUndefined()`

In these methods the parameter `ii` always refers to a table entry, which references the corresponding line of JavaScript source code; e.g. each data type can be accessed like an array. In case of an array, the access is “as supposed”, in case of a String it is character-wise, in all other cases an implicit conversion creates a new, empty array. As our runtime environment produces warnings, if implicit conversion take place, the implementation of an array access includes the statement `Log.variableTypeChangeImplicit(ii);`. In the messages table (generated by the compiler) entry `#ii` references information needed for a reasonable warning; e.g. during the execution of

```
var number = 42;
number = "Hello World";
```

the runtime environment produces the warning

```
assignment provoked a warning.
type      : variable type change by assignment
file      : C:/Users/ullrich/warning.ecs
line      : 2
details   : number = "Hello World";
```

The access functions reveal the implementation details and the mappings of the Java types.

Boolean: The corresponding Java type is boolean.

Number: A JavaScript number is mapped to double.

String: String is mapped to String.

Array: A JavaScript array is realized using the collection ArrayList<Var>.

Object: And an object in JavaScript is mapped to HashMap<String, Var>.

Function: The corresponding object to a JavaScript functor is a function pointer implementation in Java via abstract objects.

With these data types comes the necessity to use a runtime environment in the translated Java code. Whenever a variable is created or a value is assigned, a method-call is performed – thus significantly increasing the execution time of the code. However, for creating variables, a factory pattern is applied with the inherent advantage of exchangeability. This design pattern is extensively used by the “Differential Java” backend, which is described in the next section.

Concerning language constructs a wide range can be translated easily, since they have the same semantic meaning in both languages. Sometimes, there is the need to utilize temporary variables, which implicate a possible naming conflict with variable names used in the original JavaScript source code. This problem is tackled by prefixing all original JavaScript names and additionally creating unique names for temporary variables as mentioned before.

Functions: In Java, invokable routines are called methods and they are similar to, but not quite like functions in JavaScript. The runtime environment provides a class for JavaScript functions to mimic their behavior. An important property of functions in JavaScript is that they can be undefined. Therefore, when instantiating an empty function in Java, a dummy with the correct behavior is returned. Executing a function in the Java runtime environment is done by calling the execute method in the function class. In addition to function parameters, an environment reference is passed to the function in order to enable correct interaction with the immediate environment. Functions extend an abstract class called Fct defining all necessary methods:

- getID()
- getName()

- getTranslatedName()
- getAnnotations()
- getParam()
- getParams()
- execute(int ii, Var THIS, Var[] parameters)
- execute(int ii, Var THIS, Var usr_vecArray)

They reside in a *public, final* class called Function. Consequently, the function

```
function add(a, b) {
    return a + b;
}
```

gets translated to

```
@Override
public Var execute(int ii, Var THIS,
                   Var usr_a, Var usr_b) {
    try {
        if (Main.AVOID_UNREACHABLE_CODE_ERROR)
            return Op.ADD(0, usr_a, usr_b);
    } catch (EuclidesRuntimeException exp) {
        throw exp;
    } catch (RuntimeException exp) {
        Log.uncaughtException(ii);
        System.err.println(exp);
        System.exit(0);
    }
    return Factory.initUndefined();
}
```

The body of the function is embedded in a try-catch block in order to throw runtime exceptions or halt execution in case of an unhandled exception. The value undefined is returned in case a runtime exception occurs. Please note, the static constant Main.AVOID_UNREACHABLE_CODE_ERROR is always true and only needed to avoid – as it says – “unreachable code errors” thrown by Java compilers, for example, if a return-statement is followed by further statements.

Translated functions and parameters are named just like their JavaScript-counterparts (except for the usr_ prefix).

Operators: Since JavaScript data types are not mapped to native Java data types, all operators need to be recreated in the Java runtime environment as well. A total of 35 operators grouped in unary, binary and tertiary operators are available. Since each operator is applied via a method call, they can be easily exchanged. Operators are collected as methods in a *public, final* class called Op. As an example, the following operation

```
var c = 19.0 + 23.0;
```

results in

```
Variable.usr_c.assign(1, Op.ADD(0,
Factory.initNumber(19.0),
Factory.initNumber(23.0)));
```

The result of the call to Op.ADD with the two numbers as parameters is stored in a new variable, which is returned and then used as a parameter for the assignment operation.

Control Flow: Control flow statements are widely identical in both languages. One of the differences, however, is the switch-statement. For a switch-statement in Java only primitive data types are allowed, whereas JavaScript allows all types to be used, attributable to dynamic typing. In order to obtain a correct translation, the switch-statement needs to be rewritten, which is done directly in the translated code. The first step is to analyze the statement from back to front comparing each case with the switching expression. Then the result is stored in a temporary variable and the switch-statement is rebuilt in reverse order using the temporary variable as switching expression. As a result

```
switch (favoritelanguage) {
    case "Java":
        io_stdout_write("Good choice!");
        break;
    case "C":
        io_stdout_write("Bad choice");
        break;
    default:
        io_stdout_write("I have no idea");
}
```

becomes

```
int sys_42 = 0;
if (Op.EQ(9, Variable.usr_favoritelanguage,
    Factory.initString("C")).toBoolean())
    sys_42 = 1;
if (Op.EQ(10, Variable.usr_favoritelanguage,
    Factory.initString("Java")).toBoolean())
    sys_42 = 2;
switch(sys_42) {
    case 2:
        Function.usr_io_stdout_write.execute(11,
            THIS, Factory.initString("Good choice!"));
        if (Main.AVOID_UNREACHABLE_CODE_ERROR) break;
    case 1:
        Function.usr_io_stdout_write.execute(12,
            THIS, Factory.initString("Bad choice"));
        if (Main.AVOID_UNREACHABLE_CODE_ERROR) break;
    default:
        Function.usr_io_stdout_write.execute(13,
            THIS, Factory.initString("I have no idea"));
}
```

The corresponding translation in Java creates the temporary variable sys_42 for comparisons and a switch-statement in reverse order to rebuild the behavior of the JavaScript counterpart.

Once all target files containing source code are generated, they are compiled using the Java compiler included in Java Platform, Standard Edition (Java SE). The resulting class files are automatically packed into a single JAR file for easy execution. As a last step, the JAR file is digitally signed to be ready-to-use for Java Web Start. The signature information becomes part of the embedded manifest file.

B. Differential Java

Besides the previously described Java target, *Euclides* offers a Differential Java backend. Computing derivatives of functions is a necessary task in many applications of scientific computing, e.g. validating reconstruction and fitting results of laser scanned surfaces [32], [33]: In combination with variance analysis techniques, generative descriptions can be used to validate reconstructions. Detailed mesh comparisons can reveal smallest changes and damages. These analysis and documentation tasks are needed not only in the context of cultural heritage but also in engineering and manufacturing. The *Euclides* framework is used to implement generative models, whose accuracy and systematics describe the semantic properties of an object; whereas the actual object is a real-world data set (laser scan or photogrammetric reconstruction) without any additional semantic information.

This analysis task needs derivatives of the distance-based objective function as well as the embedded procedural descriptions. According to Hammer et al. [34] there are three different methods to obtain values of derivatives:

- Numerical differentiation uses difference approximations to compute approximations of the derivative values.
- Symbolic differentiation computes explicit formulas for the derivative functions by applying differentiation rules.
- Automatic differentiation also uses the well-known differentiation rules, but it propagates numerical values for the derivatives.

Automatic differentiation combines the advantages of symbolic and numerical differentiation [35]. There are two important things to mention:

- Numbers instead of symbolic formulas must be handled.
- The computation of the derivative values is done automatically together with the computation of the function value.

Automatic differentiation evaluates functions specified by algorithms or formulas. All operations are performed according to the rules of a differentiation arithmetic given by “C++ for Verified Computing” [34]. First order differentiation

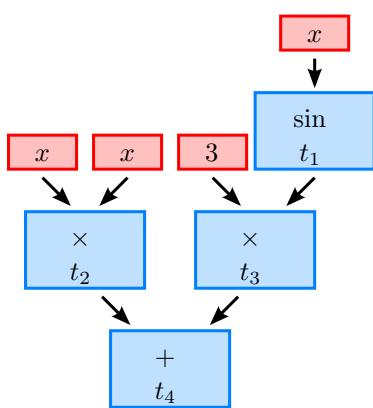


Figure 5. The evaluation of the term $x^2 + 3 \sin x$ at $x_0 = 1.3$ using differentiation arithmetic does not only return its value but also its derivative value. The computational complexity of this differentiation arithmetic (forward method) is at most a small multiple of the cost of evaluating the term itself.

arithmetic is an arithmetic for ordered pairs in the one-dimensional case: the first component contains the value $u(x)$ of the function $u : \mathbb{R} \rightarrow \mathbb{R}$ at the point $x \in \mathbb{R}$. The second component contains the value of the derivative $u'(x)$. Familiar rules of calculus are used in the second component. The operations in these definitions are operations on real numbers.

An independent variable x and the arbitrary constant c correspond to the ordered pairs

$$(x, 1) \text{ and } (c, 0), \quad (1)$$

since $\frac{dx}{dx} = 1$, and $\frac{dc}{dx} = 0$. If the independent variable x of a formula for a function $f : \mathbb{R} \rightarrow \mathbb{R}$ is replaced by $X = (x, 1)$, and if all constants are replaced by their $(c, 0)$ representation, then the evaluation of f using the rules of differentiation arithmetic gives the ordered pair

$$f(X) = f((x, 1)) \quad (2)$$

$$= (f(x), f'(x)). \quad (3)$$

For example, Figure 5 shows an AST whose evaluation at $x_0 = 1.3$ illustrates the calculation of its derivative values at intermediate subterms. For elementary functions

$$s : \mathbb{R} \rightarrow \mathbb{R} \quad (4)$$

the rules of differentiation arithmetic must be extended using the chain rule

$$s(U) = s((u, u')) \quad (5)$$

$$= (s(u), u' \cdot s'(u)). \quad (6)$$

This way the sine function is defined by

$$\sin U = \sin(u, u') \quad (7)$$

$$= (\sin u, u' \cdot \cos u). \quad (8)$$

The result of this structure and its corresponding operators is the algebra of dual numbers [36], which can be implemented in three ways:

Many programming languages offer an overloading mechanism that replaces each real number by a pair of real numbers including the differential. Each elementary operation on real numbers is overloaded, i.e., internally replaced by a new one, working on pairs of reals, that computes the value and its differential. In this way the original program is virtually unchanged.

Another approach uses source code transformation. This technique adds new variables, arrays, and data structures into the program that will hold the derivatives and the new instructions that compute them. This approach does not depend on language features such as operator overloading.

The third way to implement automatic differentiation does not modify a program or its source, but the platform (e.g. Java Virtual Machine, .Net Common Language Runtime, etc.) it runs on.

The Java differential target uses the third approach to automatically obtain derivatives. This is done by replacing variables and operators in the runtime environment, which is an easy task, since variables and operators are created using the factory pattern. The following listing shows the differences between standard and differential multiplication operator. As expected, the standard operator returns a variable initialized with the result of the multiplication operation.

```

/**
 * Binary operator multiply.
 *
 * @param ii Information index.
 * @param v1 The first operand.
 * @param v2 The second operand.
 * @return The result.
 */
public static Var MUL(int ii, Var v1, Var v2)
{
    if (!v1.getType().equals(Type.NUMBER)
        || !v2.getType().equals(Type.NUMBER))
        Log.deviantOperatorCallNoNumber(ii);

    return Factory.initNumber(
        v1.toNumber() * v2.toNumber());
}
  
```

The differential operator calculates the derivatives of the operands and stores them in an array. Then a resulting array is constructed out of the calculated derivatives and returned as a variable.

```

/**
 * Binary operator multiply.
 *
 * @param ii Information index.
 * @param v1 The first operand.
 * @param v2 The second operand.
 * @return The result.
 */
public static Var MUL(int ii, Var v1, Var v2)
{
    if (!v1.getType().equals(Type.NUMBER)
        || !v2.getType().equals(Type.NUMBER))
        Log.deviantOperatorCallNoNumber(ii);

    double[] d1 = v1.toDifferential();
    double[] d2 = v2.toDifferential();
    double[] r = Factory.differential();
    r[0] = d1[0] * d2[0];
    for(int i=1; i<r.length; i++)
        r[i] = d1[i]*d2[0] + d1[0]*d2[i];

    return Factory.initNumber(r);
}

```

C. GML

The Generative-Modeling-Language (GML) is a procedural modeling environment predominantly used in the context of Cultural Heritage [37]. The corresponding translation mechanism within *Euclides* has already been described in “Euclides – A JavaScript to PostScript Translator” and presented at the International Conference on Computational Logics, Algebras, Programming, Tools, and Benchmarking [1].

Data Types: In JavaScript each variable has a particular, dynamic type. It may be `undefined`, `boolean`, `number`, `string`, `array`, `object`, or `function`. GML also has a dynamical type system. Unfortunately, both type systems are incompatible to each other. Therefore, translating JavaScript data types to GML poses two particular problems: On the one hand, the dynamic types must be inferred at run time. On the other hand, GML’s native data types lack distinct features needed by JavaScript. GML-Strings, for example, cannot be accessed character-wise. We solved these problems by implementing JavaScript-variables as dictionaries [25] in GML. Dictionaries are objects that map unique keys to values. These dictionaries hold needed metadata and type information as well as methods, which emulate JavaScript behavior. As we will show later, we will utilize GML’s dictionaries for scoping as well.

The system translation library for GML, which every JavaScript-translated GML program defines prior to actual program code, contains the function `sys_init_data`,

which defines an anonymous data value in the sense of JavaScript data.

```

/sys_init_data {
    dict begin
    /content dict def
    content begin
        /type edef
        /value edef
        /length { value length } def
    end
    content
    end
} def

```

`sys_init_data` opens a new variable-scope by defining a new, anonymous dictionary and opening it. In this new scope, another newly created dictionary is defined by the name `content`. This content-dictionary receives three entries: `type`, `value` and the method `length`. Each entry value is taken from the top of GML’s stack. The newly created dictionary is then pushed onto the stack and the current scope is destroyed by closing the current dictionary, leaving the anonymous dictionary on the stack. In GML notation, a JavaScript-variable’s content is defined by pushing the actual value and a pre-defined constant to identify the type of the variable (such as `Types.number`, `Types.array`, etc.) onto the stack, and calling `sys_init_data`. The translator prefixes all JavaScript-identifiers with `usr_` (in order to ensure that all declarations of identifiers do not collide with predefined GML objects) and uses the following translations:

Undefined: Variables of type `undefined` result from operations that yield an `undefined` result or by declaring a variable without defining it. `var x;` leads to `x` being of type `undefined`. It is translated to

```
/usr_x Nulls.Types.undefined
Types.undefined sys_init_data def
```

Boolean: In JavaScript, boolean values are denoted by the keywords `true` and `false`. The translation simply maps these values to equivalent numerical values in GML, which interprets them. The JavaScript-statement `var x = true;` becomes

```
/usr_foo 1 Types.bool sys_init_data def
```

Number: All JavaScript numbers (including integers) are represented as 32-bit floating point values. As GML stores numbers as 32-bit floats internally as well, we simply map them to GML’s number representation. For the sake of completeness, `var x = 3.14159;` is translated to

```
/usr_x 3.14159 Types.number sys_init_data def
```

String: Although GML does support strings, they cannot be accessed character-wise. We cope with this limitation by defining strings as GML-arrays of numbers. Each number is the Unicode of the respective character. As GML allows to retrieve and to set array-elements based on indexes, this

approach meets all conditions of JavaScript-strings. The statement `var x = "Hello World";` becomes

```
/usr_x
[ 72 101 108 108 111 32 87 111 114 108 100 ]
Types.string sys_init_data def
```

Array: JavaScript arrays allow to hold data with different types, the array's contents may be mixed. This behavior is in line with GML. The JavaScript-example `var x = [true, false, "maybe"];` has a straightforward translation:

```
/usr_x [ 1 Types.bool sys_init_data
0 Types.bool sys_init_data
[109 97 121 98 101]
Types.string sys_init_data ]
Types.array sys_init_data def
```

Object: In JavaScript an object consists of key-value-pairs, e.g., `var x = {x: 1.0, y: 2.0, z: 42};` This structure is mapped to nested GML-dictionaries. The value of a variable's content is a dictionary of its own. This dictionary contains the entries corresponding to JavaScript-object's members, which are also defined as variable contents.

The example above defines a JavaScript-object of name `x` with key-value-pairs `x` to be 1, `y` to be 2, and `z` to be 42:

```
/usr_x dict begin
/obj dict def obj begin
/usr_x 1.0 Types.number sys_init_data def
/usr_y 2.0 Types.number sys_init_data def
/usr_z 42.0 Types.number sys_init_data def
end obj
Types.object sys_init_data end def
```

Opening an anonymous dictionary creates a new scope. In this scope, a dictionary is created and bound to the name `/obj`. It is then opened and its members are defined, just like anonymous variables would be. The object dictionary is then closed, put on the stack, and used to define an anonymous variable. The enclosing anonymous scoping dictionary is then closed and simply discarded.

JavaScript objects may hold functions. Our translator *Euclides* handles JavaScript object-functions like ordinary functors (next subsection) and assigns their internal name to a key-value-pair.

Function: JavaScript has first-class functions. Therefore, it is possible to assign functions to variables, which can be passed as parameters to other functions, for example. In the following example, a function `function do_nothing() {}` is declared and defined. Afterwards, the function is assigned to a variable `var x = do_nothing;`. If we abstract away from the translation of the function `do_nothing`, the statement `var x = do_nothing;` becomes:

```
/usr_do_nothing {
%% ... definition of function omitted ...
} def

/usr_x
/usr_do_nothing Types.function
sys_init_data def
```

In JavaScript, `x` can now be used as a functor, which acts the same ways as `do_nothing`. Because such functors can be reassigned, it is necessary to handle functor calls (`(x())`) differently than ordinary function calls (`(do_nothing())`). In this situation *Euclides* creates a temporary array, which contains the functor parameters and passes this array as well as the variable referencing the function name to a system function `sys_execute_var`. This system function resolves the functor and determines the referenced function, unwraps the array and performs the function call.

Functions: In GML, functions are defined using closures, such as `/my_add { add } def`. If this function `my_add` is executed, the closure `{ add }` is put onto the stack, its brackets are removed, and the content is executed.

To execute a GML function, its parameters need to be put on the stack prior to the function call: `1.0 2.0 my_add`. The resulting number `3.0` will remain on the stack. Please note, that GML functions may produce more than one result (left on the stack) at each function call. This allows to define functions with more than one result value. Following JavaScript, called functions return only one value by convention. The number and names of function parameters are known at compile time. Only functors (referenced functions stored in variables) may change at run time and cannot be checked ahead of time.

Translated functions and parameters are named just like their JavaScript-counterparts (except for their `usr_` prefix).

Scopes: As JavaScript uses a scoping mechanism different to GML, it has to be emulated. This is a rather difficult task, which has to take the following properties of JavaScript scopes into account.

- JavaScript functions may call other functions or themselves.
- Called functions may declare the same identifiers as the calling functions.
- Within functions other functions may be defined.
- Blocks might be nested inside functions, redefining symbols or declaring symbols of the same name.

The translator uses GML's dictionary mechanism to emulate JavaScript-scopes. A dictionary on the dictionary stack can be opened and it will take all subsequent assignments to GML-identifier (variables). Since only the opened dictionary is affected, this behavior is the same as the opening and closing scopes in different scoped programming languages, such as C or Java.

Thus an assignment `/x 42 def` can be put into an isolated scope by creating a dictionary (`dict`), opening

it (`begin`), performing the assignment, and closing the dictionary (`end`). The following example shows how such GML scopes can also be nested:

```
dict begin
/x 3.141 def          %% x is 3.141
dict begin
/x 4 def              %% x is 4.0
end                   %% x is 3.141
end                   %% x is unknown
```

As noted before, JavaScript supports redefinition of identifiers that were declared in a scope below the current one. Fortunately, GML exhibits just the same behavior when reading out the values of variables/keys from dictionaries of the dictionary stack. Consequently, the following example works as expected.

```
dict begin
/x 42 def
dict begin
/y x 1 add def      %% y is now 43
end
end
```

However, assignments to variables have to be handled differently in GML. The Generative Modeling Language does not distinguish between declaration and definition, any declaration must be a definition and vice versa.

The translator solves this problem. It uses a system function called `sys_def`, which is included into all translated JavaScript sources automatically. This function applies GML's `where` operator to the dictionary stack in order to find the uppermost dictionary, where the searched name is defined. The operator returns the reference to the dictionary, in which the name was found.

Control Flow for Functions: The Generative Modeling Language and all PostScript dialects lack a dedicated jump operation in control flow. Imperative functions often require the execution context to jump to a different point in the program at any time - and to return from there as well.

Fortunately, GML provides an exception mechanism. A GML exception is propagated down GML's internal execution stack until a `catch` instruction is encountered. In this way it overrides any other control structure it encounters. We use GML's exception mechanism to jump outside a function as illustrated in the following empty function skeleton:

```
/usr_foo {
dict begin
/return_issued 0 def
{ dict begin
%% ... function body omitted ...
end }
{ /return_issued 1 def }
catch

return_issued not
{ Nulls.Types.undefined
Types.undefined sys_init_data } if
```

```
end
sys_exception_return_handler
} def
```

In this empty skeleton, the function opens a new anonymous scope. Inside this scope `dict begin ... end` the local identifier `/return_issued` is set to 0. Afterwards a GML try-catch-statement `{ try_block } { catch_block }` `catch` contains the JavaScript-function implementation. In this translation, the catch block redefines `/return_issued` to 1 to indicate that a JavaScript `return` statement has been executed in the function body. JavaScript functions without any `return` statement automatically `return null` resp. in GML `Nulls.Types.undefined` `Types.undefined` `sys_init_data`. A corresponding JavaScript-return statement, e.g., `return 42;`, is translated to

```
42.0 Types.number sys_init_data end throw
```

In this example, the number `42.0` is put onto the stack. The actual function body's scope is closed `end`, and the `throw` operator is applied. The distinction of whether the end of the function body was reached by normal program flow or via a `return` statement determines, if a return value needs to be constructed (`null`) and put onto the stack.

Parameters to functions are simply put on the stack. The function body retrieves the expected number of parameters and assigns them to dictionary entries of the outer scope defined in the function translation. A complete example of a translated JavaScript-function shows the interplay of all mechanisms. The simple JavaScript-function

```
function foo(n) { return n; }
```

is translated to

```
/usr_foo {
dict begin
/usr_n edef
/return_issued 0 def
{ dict begin
usr_n
end
throw
end }
{ /return_issued 1 def }
catch

return_issued not
{ Nulls.Types.undefined
Types.undefined sys_init_data } if
end
sys_exception_return_handler
} def
```

A function call, for example `foo(3)`, yields the translation `3.0 Types.number sys_init_data usr_foo`. If we assign the function `foo` to a variable `foo_functor`, the calling convention in GML would change significantly.

```
/usr_foo_functor
/usr_foo Types.function sys_init_data def
```

is called via

```
[ 3.0 Types.number sys_init_data ]
/usr_foo_functor sys_execute_var
```

and represents the JavaScript call `foo_functor(3.0);`

Exceptions: The language JavaScript offers support for throwing exceptions as shown in the following example:

```
throw "Error: unable to read file.;"
```

Its syntax is similar to a return statement. To implement such behavior, we also use GML's exception handling mechanism. The *Euclides* translator adds a call to the predefined system function `sys_exception_return_handler` at the end of each translated function (see example above).

Throwing an exception in JavaScript translates into a global GML variable `exception_thrown` being set to 1, closing the current dictionary and calling GML's `throw`. The `sys_exception_return_handler` will check if an actual exception is being thrown, and if so, calls `throw` again. A catch-block inside a JavaScript program would set `exception_thrown` to 0.

Operators: The evaluation of expressions demands variables to be accessed. While GML provides operators that operate on their own set of types, they obviously cannot be used to access the translated/emulated JavaScript-variables. For this reason, the *Euclides* translator automatically includes a set of predefined GML functions that substitute operators defined in JavaScript.

Value Access: Performing the opposite operation to `sys_init_data`, `sys_get_value` will retrieve the data saved in a JavaScript-variable resp. its GML-dictionary. For example, to retrieve `v.value` the function `sys_get_value` is applied to `v`.

```
/sys_get_value { begin value end } def
```

Element Access: The system function `sys_get` implements string, array and object access. Applied to a string / an array `Arr` and index `k`, it will return the element `Arr[k]`. If its parameters are an object `Obj` and an attribute name, the function `sys_get` executes `Obj.name`. This may result in a value, which is put on the stack or in a function, which is called. Conforming to JavaScript, it returns JavaScript `undefined` for any requested elements that do not exist.

```
/sys_get {
    dict begin
    /idx exch def /var exch def

    var.type Types.string eq {
        %% ... handling strings ...
    } if

    var.type Types.array eq {
```

```
        %% ... handling arrays ...
    } if

    var.type Types.object eq {
        var sys_get_value idx known 0 eq {
            %% return null, if element
            %% does not exist
            Nulls.Types.undefined
            Types.undefined sys_init_data
        } if
        var sys_get_value idx known 0 ne {
            %% access element
            var sys_get_value idx get
        } if
    } if
} end
} def
```

Analogous to `sys_get`, `sys_put` inserts data into strings and arrays, or defines members of objects. If `sys_put` encounters an index `k` that is out of an array's range, the array is resized and filled with JavaScript `undefined`s.

Functors: The already mentioned routine `sys_execute_var` inspects a given variable. If it is a function, it will retrieve the array supplied to hold all parameters and execute the function. The dynamic binding of functions to variables requires to consider two situations at run time: The functor receives the correct amount of parameters for its function, or the number of parameters does not correspond to the referenced function. In the latter case, the function is not called and `null` is returned instead.

At compile time, a function is defined to expect a concrete number of parameters. This information is kept to perform parameter checks at run time. In this way, the correct number of parameters for all functors can be determined any time.

JavaScript built-in Operators: To illustrate the translation of relational, arithmetical or bit-shift operators defined by JavaScript, we discuss the equal operator `==`. It is (like all such operators) mapped to a corresponding routine `sys_eq`. Depending of the operands' types it delegates the comparison to subroutines such as `bool_eq`, `string_eq` or `array_eq` that perform the actual comparison. If the types and the values do match, `sys_eq` directly returns the JavaScript-value `true`. If types do not match, the variable is converted to the type of the respective operand, as specified by JavaScript, and then compared.

Control Flow: The JavaScript if-then-else statement corresponds one-to-one to the same GML statement. Consequently, the conditional expression is translated straightforwardly. Using the expression mapping introduced in the previous section (e.g. `sys_eq` implements the equality operator), the JavaScript statement

```
if(a == b) { c = a; } else { c = b; }
```

is translated into:

```

%% if (a==b)
usr_a usr_b sys_eq sys_get_value
{ %% then:
  dict begin {
    dict begin
      /usr_c usr_a sys_def
    end
  } exec end
}
{ %% else:
  dict begin {
    dict begin
      /usr_c usr_b sys_def
    end
  } exec end
} ifelse

```

The exec-statements (and their closures) stem from the fact that both sub-statements, the then-part and the else-part, are statement blocks { ... }. These blocks are executed within their own, new scopes.

Loops: GML supports different types of looping control structures, which have similar names to JavaScript-loops (e.g., both languages have a for-loop). However, the GML counterparts have different semantics (e.g., GML's for-loop has a fixed, finite number of iterations, which is known before execution of the loop body, whereas JavaScript-loops evaluate the stop condition during execution, which may result in endless loops). The *Euclides* translator uses the GML `loop` mechanism, which is an infinite loop that can be quit using the `exit` operator.

An important problem is that control structures such as for, while and do-while are not only controlled by the loop's stop condition, but also by JavaScript statements such as `continue` and `break` within the loop body (besides `return` and `throw` as mentioned before). The statement `break` immediately stops execution of the loop and leaves it, whereas `continue` terminates the execution of the current loop iteration and continues with the next iteration of the loop. Therefore, we translate an empty while loop `while(false) { ... }` to

```

{ /continue_called 0 def
{ 0 Types.bool sys_init_data
  sys_get_value not { exit } if
  { dict begin
    %% ... loop body omitted ...
  end
  } exec
} loop
continue_called not { exit } if
} loop

```

GML's `exit` keyword terminates the current loop. This behavior is leveraged by the *Euclides* translator to implement `break` and `continue`. The translation uses two nested loops that will run infinitely.

Prior to the begin of the inner loop `/continue_called` is set to 0. At the top of the inner loop, the loop condition is

tested. If the condition evaluates to `false`, the inner loop is exited using GML's `exit`. Otherwise a new scope is created and the loop-statement executed within that scope.

During loop iterations, there are three scenarios under which a loop can terminate:

- 1) If the loop condition is met: When the condition evaluates to `false`, the inner loop is exited. Since `continue_called` is not set to `true`, the outer loop will terminate as well.
- 2) If the loop body encounters JavaScript `break` (resp. GML `exit`): Again, the inner loop is left. `continue_called` will not be set to `true`, hence the outer loop will also terminate.
- 3) If the function returns: GML's exception throwing mechanism will unwind the stack until the catch-handler at the end of the function is encountered.

If the loop body encounters a JavaScript-`continue` statement, `continue_called` will be set to `true` and the GML `exit` command will immediately stop the inner loop. Since `continue_called` is set, execution does not leave the outer loop, however. As a consequence, `continue_called` becomes 0 again, and execution re-enters the inner infinite loop.

The do-while-statement is translated very similar to the while-statement. The only semantic differences in JavaScript are that execution will enter the loop regardless of the loop-condition and that the loop-condition is tested after loop body execution. *Euclides* translates an empty do-while-statement `do { ... } while(false)` as follows:

```

{ /continue_called 0 def
{ { dict begin
    %% ... loop body omitted ...
  end
  } exec
0 Types.bool sys_init_data
  sys_get_value not { exit } if
} loop
continue_called not { exit } if
0 Types.bool sys_init_data
pop
} loop

```

Due to a semantic difference of JavaScript `continue` in do-while-loops, this statement needs to be handled differently. If `continue` is encountered, the loop condition must still execute before the loop body is re-entered, because side effects inside the loop condition may occur (such as incrementing a counter). *Euclides* handles this problem by executing the condition expression a second time in the outer loop. Since expressions always return values, any value resulting from the loop-expression has to be popped off the stack.

Although GML has a `for` operator, it is semantically incompatible with JavaScript's one. Its increment is a constant number, and so is the limit. In JavaScript, both increment and limit must be evaluated at each loop body

execution. Therefore, we translate `for` just like the previous constructs by two nested loops with the increment condition repeated in outer loop (due to `continue` semantics). Finally, *Euclides* translates the JavaScript statement `for (var i=0; i<1; i++) { }` to GML via

```

dict begin
%% initialization (i=0)
/usr_i 0.0 Types.number sys_init_data def
{ /continue_called 0 def
  { %% condition (i<1)
    usr_i 1.0 Types.number
    sys_init_data sys_lt
    sys_get_value not { exit } if
    { dict begin
      %% ... loop body ...
      end
    } exec
    %% increment (i++)
    usr_i
      usr_i 1 Types.number
      sys_init_data sys_add
    /usr_i sys_edef
    pop
  } loop
  continue_called not { exit } if
  %% increment again (i++)
  usr_i
    usr_i 1 Types.number
    sys_init_data sys_add
  /usr_i sys_edef
  pop
} loop
end

```

In JavaScript, the following `for-in` statement `for(var x in array) statement;` is semantically equivalent to:

```

for(var i = 0; i < array.length; i++) {
  var x=array[i]; statement;
}

```

This construction loops over the elements of an array provides the loop body with a variable holding the current element.

Selection Control Statement: The translation of the JavaScript `switch` statement poses several difficulties:

- If a case condition is met, execution can “fall through” till the next `break` is encountered.
- If a `break` is encountered, the currently executed `switch` statement must be terminated.
- Of course, `switch` statements may be nested.

To develop a semantically consistent solution, we did not want to alter the translation of JavaScript-`break` inside `switch` statements (compared to loops). We solve the problem of breaking outside the `switch` statement by implementing it as a loop that is run exactly once. In GML it reads like `1 { loop_instructions } repeat`. This way our translation of `break` shows semantically correct behavior,

it terminates the loop. Consider the following JavaScript-program:

```

var x = 0, y = 0;

function bar() { return 3; }

function foo(i) {
  switch(i) {
    case 0:
    case 1:
    case 2: x = 1;
    case 4: x = 3;
    case bar(): x = 2; break;
    default: y = 5;
  }
}

```

The function `foo` will be translated to:

```

/usr_foo
{ dict begin
  /usr_i edef
  /return_issued 0 def
  { dict begin
    /switch_cnd_met1 0 def
    1 { usr_i 0.0
      Types.number sys_init_data
      sys_eq sys_getvalue
      switch_cnd_met1 1 eq or {
        /switch_cnd_met1 1 def
      } if

      usr_i
      1.0 Types.number sys_init_data
      sys_eq sys_getvalue
      switch_cnd_met1 1 eq or {
        /switch_cnd_met1 1 def
      } if

      usr_i
      2.0 Types.number sys_init_data
      sys_eq sys_getvalue
      switch_cnd_met1 1 eq or {
        /switch_cnd_met1 1 def
        %% x = 1;
        /usr_x 1.0 Types.number
          sys_init_data sys_def
      } if

      usr_i
      4.0 Types.number sys_init_data
      sys_eq sys_getvalue
      switch_cnd_met1 1 eq or {
        /switch_cnd_met1 1 def
        %% x = 3;
        /usr_x 3.0 Types.number
        sys_init_data sys_def
      } if

      usr_i usr_bar
    } if
  } if
} if

```

```

sys_eq sys_getvalue
switch_cnd_met1 1 eq or {
    /switch_cnd_met1 1 def
    %% x = 2;
    /usr_x 2.0 Types.number
    sys_init_data sys_def
    exit
} if
%% y = 5;
/usr_y 5.0 Types.number
sys_init_data sys_def
} repeat
currentdict /switch_cnd_met1 undef end
}
{/return_issued 1 def } catch

return_issued not {
    Nulls.Types.undefined
    Types.undefined sys_init_data
} if
end
sys_exception_return_handler
} def
}

```

This example shows that we introduce an internal variable `/switch_cnd_metX` for traversing the case statements. As soon as a case statement condition is met, `/switch_cnd_metX` is set to `true`, leading execution into every encountered case statement.

The *Euclides* translator takes into account that switch statements may be nested. As it traverses the AST, it keeps book of all internal variable to ensure a unique name (`switch_cnd_met1`, `switch_cnd_met2`, ..., `switch_cnd_metN`).

The example translation shows that for `foo(3)` the cases 0, 1, 2, 4 and 3 (= `bar()`) will only execute case 3, where the `1 { } repeat` statement will be broken out of with the GML `exit` operator. The default block will be executed in any case if execution is still inside the `repeat` statement, no further state is checked for `default`.

The JavaScript to PostScript translation target reduces the inhibition threshold of the GML significantly. Even advanced GML users, who already know how to program in PostScript style, can use *Euclides* to translate algorithms, which are often presented in a imperative, procedural (pseudo-code) style [38].

V. CONCLUSION AND FUTURE WORK

The correct translation of control flow structures to various target platforms is a non-trivial task. For example, due to the fact that there is no concept of `goto` in the PostScript language and its dialects, the main challenge is the complete translation of JavaScript into a PostScript dialect including all control flow statements. To the best of our knowledge, this is the first complete translator. Other projects (PdB by Arthur van Hoff, pas2ps by Dulith Herath and Dirk Jagdmann) do not support e.g., return statements.

The main contribution is the meta-modeler concept, which allows a user to export generative models to other platforms without losing its main feature the procedural paradigm. It is well suited for procedural modeling, has a beginnerfriendly syntax and is able to generate and export procedural code for various, different generative modeling or rendering engines. The source code does not need to be interpreted or unfolded, it is translated. Therefore, it can still be a very compact representation of a complex model.

The target audience of this approach consists of beginners and intermediate learners of procedural modeling techniques and addresses application domain experts (e.g., archaeologists in a cultural heritage project) who are seldom computer scientists. These experts are needed to tap the full potential of generative techniques.

The current IDE only offers basic functionality and some convenience when creating, editing or translating source code. For further improvements, we envisage using a Swing-based IDE framework like the NetBeans Platform, which offers a modular approach for creating rich applications. In the backend, further optimizations concerning the performance of the generated code are planned - e.g., more direct mapping onto native data types. Additional target languages would extend the application field of the framework.

The *Euclides* modeler is available in version 2.0 and can be downloaded at: <http://www.cgv.tugraz.at/euclides>.

ACKNOWLEDGMENT

We would like to thank Richard Bubel for his valuable support on ANTLR and the JS grammar. In addition, the authors gratefully acknowledge the generous support from the European Commission for the integrated project 3D-COFORM (www.3Dcoform.eu) under grant number FP7 ICT 231809, from the Austrian Research Promotion Agency (FFG) for the research project METADESIGNER, grant number 820925/18236, as well as from the German Research Foundation (DFG) for the research project PROBADO under grant INST 9055/1-1 (www.probado.de).

REFERENCES

- [1] M. Strobl, C. Schinko, T. Ullrich, and D. W. Fellner, "Euclides – A JavaScript to PostScript Translator," *Proceedings of the International Conference on Computational Logics, Algebras, Programming, Tools, and Benchmarking (Computation Tools)*, vol. 1, pp. 14–21, 2010.
- [2] D. Brutzman, "The virtual reality modeling language and Java," *Communications of the ACM*, vol. 41, no. 6, pp. 57 – 64, 1998.
- [3] J. Behr, P. Dähne, Y. Jung, and S. Webel, "Beyond the Web Browser – X3D and Immersive VR," *IEEE Virtual Reality Tutorial and Workshop Proceedings*, vol. 28, pp. 5–9, 2007.

- [4] F. Breuel, R. Berndt, T. Ullrich, E. Eggeling, and D. W. Fellner, "Mate in 3D – Publishing Interactive Content in PDF3D," *Publishing in the Networked World: Transforming the Nature of Communication, Proceedings of the International Conference on Electronic Publishing*, vol. 15, pp. 110–119, 2011.
- [5] M. Di Benedetto, F. Ponchio, F. Ganovelli, and R. Scopigno, "SpiderGL: a JavaScript 3D graphics library for next-generation WWW," *Proceedings of the 15th International Conference on Web 3D Technology*, vol. 15, pp. 165–174, 2010.
- [6] J. K. Ousterhout, "Scripting: Higher Level Programming for the 21st Century," *IEEE Computer Magazine*, vol. 31, no. 3, pp. 23–30, 1998.
- [7] R. B. OpenGL Architecture, *OpenGL Reference Manual*, R. B. OpenGL Architecture, Ed. Addison-Wesley Publishing Company, 1993.
- [8] NVidia, "NVIDIA CUDA C Programming Guide."
- [9] D. Reiners, G. Voss, and J. Behr, "OpenSG: Basic concepts," *Proceedings of OpenSG Symposium 2002*, vol. 1, pp. 1–7, 2002.
- [10] G. Voß, J. Behr, D. Reiners, and M. Roth, "A multi-thread safe foundation for scene graphs and its extension to clusters," *Proceedings of the Fourth Eurographics Workshop on Parallel Graphics and Visualization*, vol. 4, pp. 33–37, 2002.
- [11] B. Eckel, *Thinking in C++: Introduction to Standard C++*, Practical Programming, B. Eckel, Ed. Prentice Hall, 2003.
- [12] T. Ullrich, C. Schinko, and D. W. Fellner, "Procedural Modeling in Theory and Practice," *Poster Proceedings of the 18th WSCG International Conference on Computer Graphics, Visualization and Computer Vision*, vol. 18, pp. 5–8, 2010.
- [13] R. Berndt, G. Buchgraber, S. Havemann, V. Settgast, and D. W. Fellner, "A publishing workflow for cultural heritage artifacts from 3d-reconstruction to internet presentation," in *Digital Heritage. Third International Conference, EuroMed 2010*, M. Ioannides, D. W. Fellner, A. Georgopoulos, and D. Hadjimitsis, Eds., vol. 6436. Springer, 2010, pp. 166–178, doi:10.1007/978-3-642-16873-413.
- [14] M. Strobl, R. Berndt, V. Settgast, S. Havemann, and D. W. Fellner, "Publishing 3D Content as PDF in Cultural Heritage," *Proceedings of the 10th International Symposium on Virtual Reality, Archaeology and Intelligent Cultural Heritage (VAST)*, vol. 6, pp. 117–124, 2009.
- [15] V. Settgast, T. Ullrich, and D. W. Fellner, "Information Technology for Cultural Heritage," *IEEE Potentials*, vol. 26, no. 4, pp. 38–43, 2007.
- [16] C. Schinko, M. Strobl, T. Ullrich, and D. W. Fellner, "Modeling Procedural Knowledge – a generative modeler for cultural heritage," *Proceedings of EUROMED 2010 - Lecture Notes on Computer Science*, vol. 6436, pp. 153–165, 2010.
- [17] P. Prusinkiewicz and A. Lindenmayer, *The Algorithmic Beauty of Plants*, P. Prusinkiewicz and A. Lindenmayer, Eds. Springer-Verlag, 1990.
- [18] Y. Parish and P. Mueller, "Procedural Modeling of Cities," *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, vol. 28, pp. 301–308, 2001.
- [19] P. Müller, G. Zeng, P. Wonka, and L. Van Gool, "Image-based Procedural Modeling of Facades," *ACM Transactions on Graphics*, vol. 28, no. 3, pp. 1–9, 2007.
- [20] M. Lipp, P. Wonka, and M. Wimmer, "Interactive Visual Editing of Grammars for Procedural Architecture," *ACM Transactions on Graphics*, vol. 27, no. 3, pp. 1–10, 2008.
- [21] P. Müller, P. Wonka, S. Haegler, U. Andreas, and L. Van Gool, "Procedural Modeling of Buildings," *Proceedings of 2006 ACM Siggraph*, vol. 25, no. 3, pp. 614–623, 2006.
- [22] B. Lintermann and O. Deussen, "A Modelling Method and User Interface for Creating Plants," *Computer Graphics Forum*, vol. 17, no. 1, pp. 73–82, 1998.
- [23] B. Ganster and R. Klein, "An Integrated Framework for Procedural Modeling," *Proceedings of Spring Conference on Computer Graphics 2007 (SCCG 2007)*, vol. 23, pp. 150–157, 2007.
- [24] D. Finkenzeller, "Detailed Building Facades," *IEEE Computer Graphics and Applications*, vol. 28, no. 3, pp. 58–66, 2008.
- [25] S. Havemann, "Generative Mesh Modeling," *PhD-Thesis, Technische Universität Braunschweig, Germany*, vol. 1, pp. 1–303, 2005.
- [26] E. Mendez, G. Schall, S. Havemann, D. W. Fellner, D. Schmalstieg, and S. Junghanns, "Generating Semantic 3D Models of Underground Infrastructure," *IEEE Computer Graphics and Applications*, vol. 28, pp. 48–57, 2008.
- [27] C. Reas and B. Fry, *Processing: A Programming Handbook for Visual Designers and Artists*. MIT Press, 2007.
- [28] S. Davidson, "Grasshopper- generative modeling for rhino," online: <http://www.grasshopper3d.com/>, 2011.
- [29] D. Flanagan, *JavaScript. The Definitive Guide*, 5th ed. O'Reilly Media, 2006.
- [30] T. Parr, *The Definite ANTLR Reference – Building Domain-Specific Languages*, T. Parr, Ed. The Pragmatic Bookshelf, Raleigh, 2007.
- [31] T. Ullrich, U. Krispel, and D. W. Fellner, "Compilation of Procedural Models," *Proceeding of the 13th International Conference on 3D Web Technology*, vol. 13, pp. 75–81, 2008.
- [32] C. Schinko, T. Ullrich, T. Schiffer, and D. W. Fellner, "Variance Analysis and Comparison in Computer-Aided Design," *Proceedings of the International Workshop on 3D Virtual Reconstruction and Visualization of Complex Architectures*, vol. XXXVIII-5/W16, pp. 3B21–25, 2011.
- [33] T. Ullrich and D. W. Fellner, "Generative Object Definition and Semantic Recognition," *Proceedings of the Eurographics Workshop on 3D Object Retrieval*, vol. 4, pp. 1–8, 2011.

- [34] R. Hammer, M. Hocks, U. Kulisch, and D. Ratz, *C++ Toolbox for Verified Computing*, R. Hammer, M. Hocks, U. Kulisch, and D. Ratz, Eds. Springer, 1997.
- [35] A. Griewank and A. Walther, *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*, A. Griewank and A. Walther, Eds. SIAM, 2008.
- [36] M. L. Keler, "On the Theory of Screws and the Dual Method," *Proceedings of A Symposium Commemorating the Legacy, Works, and Life of Sir Robert Stawell Ball Upon the 100th Anniversary of "A Treatise on the Theory of Screws"*, vol. 1, pp. 1–12, 2000.
- [37] S. Havemann and D. W. Fellner, "Generative Parametric Design of Gothic Window Tracery," *Proceedings of the 5th International Symposium on Virtual Reality, Archeology, and Cultural Heritage*, vol. 1, pp. 193–201, 2004.
- [38] T. H. Cormen, C. Stein, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, T. H. Cormen, C. Stein, C. E. Leiserson, and R. L. Rivest, Eds. B&T, 2001.

Simulation and Test-Case Generation for PVS Specifications of Control Logics

Cinzia Bernardeschi, Luca Cassano, Andrea Domenici
 Department of Information Engineering
 University of Pisa, Italy
 {c.bernardeschi, l.cassano, a.domenici}@ing.unipi.it

Paolo Masci
 School of Electronic Engineering and Computer Science
 Queen Mary University of London, UK
 paolo.masci@eecs.qmul.ac.uk

Abstract—We describe a framework for the simulation of control logics specified in the higher-order logic of the *Prototype Verification System*. The framework offers a library of predefined modules, a method for the composition of more complex modules, and an event-driven simulation engine. A developer defines a system architecture by composing its model out of library modules, possibly introducing new module definitions, and simulates the behaviour of the system model by providing its input waveforms, which are given as functions from time to logic levels. The generation of simulation scenarios (test cases) can be automated by using parametric waveforms that can be instantiated through universal and existential quantifiers. We demonstrate the simulation capabilities of our framework on two simple case studies from a nuclear power plant application. The main feature of this approach is that our formal specifications are executable. Moreover, once the simulation experiments give developers sufficient confidence in the correctness of the specification, the logic models can serve as the basis for its formal verification.

Keywords-PVS; simulation; validation; test-case generation; control logics.

I. INTRODUCTION AND MOTIVATION

Control systems combine real-time requirements and non-trivial control tasks whose failure may compromise safety. Subtle design faults, difficult to avoid and tolerate, and the possibility of failures caused by the occurrence of non-obvious combinations of events, make such systems hard to certify with respect to safety requirements.

The use of formal methods is increasingly being required by international standards and guidelines for the development of safety critical digital control systems. Formal methods are in fact recognised as a fault avoidance technique that can increase dependability by removing errors at the requirements, specification and design stages of development. In this paper, we present a methodology (introduced in [1]) for the simulation of control logics, formally specified in the higher-order logic of the *Prototype Verification System (PVS)* [2], a specification and verification system that combines an expressive specification language with an interactive theorem prover. Thus, the same model can be used both for simulation and formal verification of system properties.

Formal methods are highly recommended by such standards as the EN 50128:2001 European Standard [3] in the requirements specification and in the design and the validation of

railway control and protection systems, and the IAEA NS-G-1.1 Standard [4] in digitalised instrumentation and control systems in nuclear power plants.

On the other hand, verification and validation (V&V) of embedded systems relies heavily, and often exclusively, on simulation and testing. In particular, simulation is often the only V&V tool in the development of ASIC- and FPGA-based hardware. A typical development process for such systems involves creating a block-diagram model through a CAD tool that generates a model of the hardware at successive levels of detail, and each of these intermediate models is simulated. Alternatively, the initial model may be expressed in a hardware description language such as Verilog [5] or VHDL [6].

A rigorous development process would benefit from the combined application of formal verification, simulation, and testing. In particular, simulation would be a means to validate specifications against requirements. However, verification tools (such as theorem provers and model checkers) and simulation tools use different languages, and few designers are versed in the use of both kinds of tools.

The work presented in this paper is focused on the validation of high level specifications of control logics, relying on executable formal specifications. We note that executable specifications are more commonly based on process algebras or state machine formalisms that are more amenable to computer execution than logic-based formalisms, but they suffer the problem of state explosion [7].

It is assumed that the development process of a control system starts from a specification expressed as function block diagrams. This specification can be translated into a higher-order logic theory that guides the execution of a simulator. When the simulation results make developers confident that their specifications express the intended system behaviour, a more detailed and formal analysis of its properties may be done by theorem proving.

In function block diagrams, each block represents some operation on digital or analog signals. Such operations include, for example, Boolean functions, comparison, voting, integration and differentiation. Functional blocks may be implemented in many ways: a single functional block may correspond to one or more hardware modules, a group of blocks may be implemented in a single hardware module, and a block or group may be implemented in software executed

by some programmable device. This work addresses systems where only digital (i.e., discrete-valued) signals are present.

We have developed a library of purely logic specifications for typical control system components, a methodology to combine them into more complex systems, and a simulation engine capable of animating the formal specifications with the PVS ground evaluator. The library comprises definitions for basic concepts, such as time, signals, and events. The simulation framework also enables test cases (input data to the simulations) to be automatically generated by using parametric waveforms that can be instantiated through universal and existential quantifiers.

The paper is organised as follows. Section II reports related work on formal verification of digital control systems. We introduce the PVS system in Section III, then we describe the theories for the logical specification of signals and control components (Section IV) and the theory defining the simulator including a theory for the events associated to signals (Section V). In Section VI, we describe two simple case studies from the field of control logics for nuclear power plants (NPPs), and, finally, the conclusion is found in Section VII.

II. RELATED WORK

The last few years have seen a continuous increase in usage of digital components in safety critical control systems. Digital control systems are flexible and enable sophisticated control schemas to be realized. However, these systems are complex and call for advanced tools and techniques to ensure compliance with safety requirements. A few examples in the literature point out to the difficulties in anticipating all risk situations and to the fact that apparently harmless events (such as small unforeseen changes in a sequence of operations) may lead to catastrophic consequences. These reasons motivate the introduction of formal methods in the development process of control systems as early as their first phases (as acknowledged by international standards). Such methods afford a precise representation of control schemas and make it possible to reason on control systems properties in a rigorous manner. The application of these methods, however, must deal with the problem of complexity of the systems to be analysed and is therefore an advanced research topic with interesting theoretical implications and relevant practical advantages.

In many research works, digital control system specifications are analyzed with logical-mathematical methods. Two lines of research emerge from these works, addressing methods based on model checking [8] and theorem proving [9], respectively. Model checking relies on generating a state model of system behavior. Properties expressed in temporal logic are automatically verified by model-checking algorithms. Theorem proving relies on a logic language and a collection of inference rules specific of each language. Verification is done by proofs assisted by a theorem proving tool that can apply inference rules in an entirely or partially automatic manner.

Several works have explored the use of model checkers and theorem provers in the field of instrumentation and control. For instance, Krämer et al. [10] used the Isabelle/HOL

theorem prover for modelling and verifying programs for programmable logic controllers. They demonstrated the utility of using formal methods on such systems, and in particular they argued that the sheer exercise of formalising system descriptions given with graphical languages, such as function block diagrams, is able to point out incomplete information about functionalities, ambiguities, contradictions and design flaws.

Wan et al. [11] used the Coq theorem prover for modelling and verifying programs for programmable logic controllers with timers. They addressed the problem of reasoning on specifications that involve timers, and they propose a set of axioms suitable to ease the modelling of timers at different levels of abstraction.

Jee et al. [12] translated function block diagrams into semantically equivalent Verilog programs that can be checked with the SMV model checker, and they implemented also a visualisation tool for animating the specifications.

Various works, like Vyatkin and Hanisch [13], and Missal et al. [14], translated function block diagrams into Signal Net Systems, a generalisation of Petri Nets, and then used ad hoc analysis tools for analysing properties of interest on the Signal Net System specification, such as reachability of dangerous states and validation of arbitrary input/output specifications.

VHDL and Verilog are commonly used for logical circuits design. Their key advantage is that they allow the behaviour of a system to be modeled and simulated before synthesis tools translate the design into real hardware. The problem of formal verification of VHDL designs is dealt with in [15] where the behaviour of a VHDL design is specified with temporal logic formulas and a model checker is applied for the verification of the design. In [16], a language to design circuits and prove properties in the Nqthm theorem prover [17] is shown. The language can be translated to a subset of VHDL.

Jain et al. [18] verified circuits described in Verilog with a model checking and predicate abstraction technique, and developed a model checking tool, VCEGAR, suitable to verify safety requirements of control system specified in Verilog [19].

The model checking analysis of complex control systems suffers of the state-space explosion problem, thus requiring abstraction techniques where verification is performed on a set of abstract states. Theorem provers are fundamental in this application field, even if this kind of tools requires specific competence of the control designer and verification is semi-automatic.

In our work, on the other hand, we have investigated the possibility of integrating an event-based simulation environment into a theorem proving system: simulations give designers an intuitive and effective way for investigating the behaviour of a system through test cases; theorem proving enables analysts to explore *all* possible behaviours of the system, which is essential in safety critical domains for detecting design errors in advance.

PVS is currently one of the most popular and powerful theorem provers, that has been used for formal reasoning in several application domains [20]. In particular, it has been

used in various works to specify and verify hardware systems, e.g., in [21][22][23]. Other application fields include fault tolerant systems [24], wireless sensor network protocols [25], and distributed cognition systems [26].

With our approach, the formal specifications are executable and they can be simulated with the ground evaluator of PVS. This way, once the simulation experiments give developers sufficient confidence in the correctness of the specification, the same PVS models can serve as the basis for the formal verification of properties in the theorem prover of PVS. It is known that a large share of defects in computing systems stem from errors in the formulation of specifications [27].

III. PVS AND PVSIO

The distinguishing characteristics of PVS [2] are its expressive specification language and its powerful theorem prover.

The PVS specification language builds on classical typed higher-order logic with the usual base types, `bool`, `nat`, `integer`, `real`, among others, and the function type constructor (e.g., type $[A \rightarrow B]$ is the set of functions from set A to set B). Predicates are functions with range type `bool`. The type system of PVS also includes record types, dependent types, and abstract data types.

PVS specifications are packaged as *theories* that can be parametric in types and constants. A collection of built-in (*prelude*) theories and loadable libraries provide standard specifications and proved facts for a large number of theories. A theory can use the definitions and theorems of another theory by *importing* it.

For instance, consider the following theory execution:

```
execution: THEORY
BEGIN
  State: TYPE
  tf : VAR [State -> State]
  execute(n_steps: nat)(tf):
    RECURSIVE [State -> State] =
      LAMBDA (s: State):
        IF n_steps = 0
        THEN s
        ELSE
          LET s_prime = tf(s) IN
          execute(n_steps - 1)(tf)(s_prime)
        ENDIF
        MEASURE n
END execution
```

The theory defines a `State` and a (higher-order) function `execute` that recursively applies `n_steps` of a state-transition function `tf`, that is provided as a parameter. As all functions in PVS must be total, the termination of the recursion has to be demonstrated; the `MEASURE` part provides the information to the typechecker and prover to ensure this. Thus, the `execution` theory provides a generic mechanism to describe the execution of a system, that can subsequently be used for simulation.

The PVS environment has an automated theorem prover that provides a collection of powerful primitive inference procedures that are applied interactively under user guidance within a sequent calculus framework. The primitive inferences include propositional and quantifier rules, induction, rewriting,

simplification using decision procedures for equality and linear arithmetic, data and predicate abstraction [28].

Although PVS offers a very expressive specification language, a large subset of the language is actually executable: all ground expressions of ground type are executable; the only fragments of the language that are not executable are uninterpreted functions, quantification over infinite domains, free variables, and equalities between higher-order terms. However, the evaluation is nonstrict, and expressions may be executed even if they contain unexecutable subexpressions.

PVS includes a *ground evaluator* [29] that can be used to evaluate, test, and animate PVS specifications by executing them on concrete data. The core of the ground evaluator is a translator that compiles executable PVS expressions into Common Lisp code. The translation is performed lazily, i.e., the translation of an expression happens only when its value is actually required. The ground evaluator also consists of an evaluation environment, which is an interactive *read-eval-print* loop that allows the user to input expressions, and returns the result of their evaluation.

The techniques used in the ground evaluator to associate Lisp programs with PVS functions are also available to the PVS user, who can provide pieces of Lisp code (called *semantic attachments*) and attach them to PVS symbols. This mechanism is useful to allow expressions that involve unexecutable constructs, such as uninterpreted functions, to be handled by the evaluator, by associating them with a suitable implementation.

Using this mechanism, the *PVSio* package [30] extends the ground evaluator with a predefined library of imperative programming language features such as side effects, unbounded loops, and input/output operations, and also provides a high-level interface for writing user-defined semantic attachments. Thus, PVS specifications can be conveniently animated within the *read-eval-print-loop* of the ground evaluator.

In our framework, we exploit the expressiveness of the PVS specification language for enabling a natural mapping between higher order logic specifications and systems models described as function block diagrams. We employ the mechanisms provided by *PVSio* for implementing a customisable simulation environment suitable to animate the model of the system expressed in higher order logic.

IV. MODELLING CONTROL LOGICS

In this section, we describe the PVS theories developed to formally model control logics. We start with the PVS theories that model time, logic levels, signals, and basic operations on signals. Then, we introduce samples of the library for the basic digital modules of a system, such as logic gates and timers. Finally, we show how to build complex components out of basic elements. The developed theories are executable: definitions always use interpreted types and quantification is always performed over bounded types. In the following sections, only the `time_th` theory will be shown in a syntactically complete form; only some relevant fragments

of PVS code will be shown in the rest of the paper for the other theories.

A. Time

Time is modelled as a variable ranging over the continuous domain of real numbers. Theory `time_th` (shown below) contains the type definition of time (modelled as ranging over the continuous domain of real numbers) and time interval. In the theory, `time` represents the relative time with respect to an initial time. The initial time is the zero reference point: negative time values represent time instants preceding the initial time, while positive time values represent time instants following the initial time. Type `interval` models time intervals with a non-negative real number (instantaneous time intervals have duration zero).

```
time_th: THEORY
BEGIN
  time: TYPE = real
  interval: TYPE = {t: time | t >= 0}
END time_th
```

B. Logic Levels

The logic levels of hardware circuits correspond, in the real world, to voltage or current levels. Besides the classical zero and one values, additional levels are needed to model *unknown* values and *high impedance*. Unknown values are useful to model the logic level when the digital circuit is powered up, while high impedance is useful to represent open circuits or mis-wiring situations (e.g., the designer forgets to wire a port of the digital circuit).

Theory `logic_levels_th` provides the definitions of the logic levels and of the logical operators over the four-valued logic. In the theory, logic levels are modelled with natural numbers (each level is associated with a unique number), and each level is associated with a mnemonic name and a recogniser predicate (denoted by the mnemonic name followed by a question mark symbol). In the following fragment, we show the definitions of types and constants, and the definition of the basic logical operator `LAND` over the four-valued logic. Other definitions are in Appendix A.

```
logic_levels_th: THEORY
BEGIN
  %-- logic level (type definition)
  logic_level: TYPE = below(4)

  %-- names of logic levels
  zero: logic_level = 0
  one: logic_level = 1
  Z: logic_level = 2    %-- high impedance
  U: logic_level = 3    %-- unknown

  %-- logical AND in a four-valued logic
  LAND(v1, v2: logic_level): logic_level =
    IF one?(v1) AND one?(v2) THEN one
    ELSIF zero?(v1) OR zero?(v2) THEN zero
    ELSE U ENDIF

  % ...
END logic_levels_th
```

C. Signals

A signal describes the variation of a logic level over time, and we represent signals as functions from the domain of time to logic levels. Signal transitions are specified pointwise, by comparing the logical level of the signals at two closely spaced time points. The spacing between time points corresponds to the *time resolution* of the digital circuit, i.e., the minimum amount of time required by the components in the circuit for detecting two observable variations of a signal. This allows us to simplify the definition of signals transitions, and also to define executable functions for detecting signals transitions over the continuous time domain. Note that the generality of properties proved on specifications involving the concept of time resolution does not affect the generality of the proof, because the actual value of the time resolution can be left unspecified (i.e., any value is possible) when doing the proof.

Theory `signals_th` contains, besides the definition of `signal`, the symbolic constant for time resolution, `tres`, which models the minimum time between two observable variations of a signal, and the definitions of utility functions to shift a signal in time (`time_shift`) and to build periodic signals (`periodic`).

Basic signals provided in the theory are: `constval`, a constant logical level; `step`, a signal that goes from zero to one at time τ ; `pulse`, a signal that is one only in the time interval $[\tau, \tau + d]$, where d is the interval size.

Some useful predicates on signals are defined, such as `rising_edge?`, used to detect if a signal s has a rising edge at time τ . Logical operations on signals are defined (`sOR`, `sAND`, `sNOT`), that apply the operator to the values of signals at each given time. Sample definitions of this theory follow. More definitions are in Appendix A.

```
signals_th: THEORY
BEGIN
  IMPORTING time_th, logic_levels_th
  %-- signal (type definition)
  signal: TYPE = [time -> logic_level]

  %-- symbolic constant of the minimum time
  % between two observable variations in a signal
  tres: posreal

  %-- definition of basic waveforms
  step(tau: time): signal =
    LAMBDA (t: time):
      IF t >= tau THEN one ELSE zero ENDIF
    %

  %-- time shift of the signal
  time_shift(s: signal, offset: time): signal =
    LAMBDA (t: time): s(t - offset)

  %-- logical operators in a four-valued logic
  sAND(s1, s2: signal): signal =
    LAMBDA (t: time):
      IF one?(s1(t)) AND one?(s2(t))
        THEN one
      ELSIF zero?(s1(t)) OR zero?(s2(t))
        THEN zero
      ELSE U ENDIF

  %
END signals_th
```

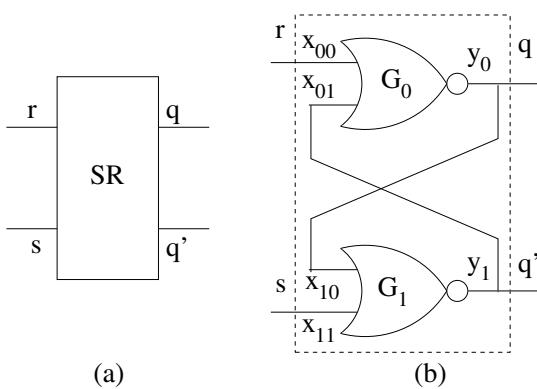


Fig. 1. An SR flip-flop.

D. Digital Modules

In our framework, a control logic is a *composite digital module*, obtained by connecting *basic digital modules*. Digital modules are characterised by a set of *ports*, a *state*, and a *transition function*.

Ports are abstractions of the terminals of physical devices. Each port is identified by its *category* (one of *input*, *output*, *internal*) and its *port number* within the category. Basic modules have only input and output ports, whereas composite modules also have internal ports. In a composite module, the input and output ports are its externally visible terminals, and its internal ports are the ports of the (basic) component modules that are not externally visible. For example, a NOR gate is modelled as a module with two input ports, one output port, and no internal ports. Another example is an SR flip-flop, which can be modelled either as a basic module (Figure 1(a)) with two input ports, two output ports and no internal ports, or as a composite module built from two NOR gates. In this case, the resulting system is shown in Figure 1(b), where ports x_{00} of gate G_0 and x_{11} of gate G_1 are input ports, ports y_0 of G_0 and y_1 of G_1 are output ports, and ports x_{01} and x_{10} are internal ports.

The *state* of a module is defined as the set of signals (i.e., functions of time modelling waveforms) applied to, or generated by, the module at a given time. Defining the state as a set of time functions instead of instantaneous values makes it possible to define the behaviour of some modules in terms of properties of such functions, thus allowing for better expressiveness. As an example, we may consider the specification of a timer, whose output depends on the *shape* (namely, the presence of a rising edge) of its input signal, along with the current value of the output:

```
timerM(d: posreal): basic_digital_module(1, 1) =
  LAMBDA (t: time):
    LAMBDA (s: state(1, 1)):
      IF rising_edge?(port0(input(s)), t) AND
        zero?(port0(output(s)), t)
      THEN s
        WITH [output := ports(pulse(t + delay, d))]
      ELSE s
      ENDIF
```

With this approach, a state transition occurs when a signal on a port is replaced by a different one. For example, let us consider the timer defined above, `timerM`. Let us suppose that the initial signals on the output port of the timer is a constant logical zero (`constval(zero)`) and that the input signal is a step, with rising edge at $t = t^*$ (`step(t_star)`). As long as $t < t^*$, the output signal remains a constant logical zero, because the condition expressed in `timerM` is false. When $t = t^*$, the condition in `timerM` becomes true, and the state changes: the constant logical zero signal on the output port is replaced with a pulse signal of duration d starting after a propagation latency (`pulse(t + delay, d)`). As a signal is formally defined over the whole time axis, all signals in the context of a given state are meant to be ‘sliced’ to the time interval in which the state holds. In the previous example, the temporal evolution of the timer defines two states, each of which is associated to a validity interval: the first state is characterised by a `step(t_star)` on the input port, a `constval(zero)` on the output port, and is valid in the time interval $[0, t^*)$; the second state has `step(t_star)` on the input port, `pulse(t_star + delay, d)` on the output port, and is valid in the time interval $[t^*, +\infty)$.

The *transition function* specifies how the state changes according to a module’s functionality. Theory `digital_modules_th` contains type definitions for the state of a digital module (`state`) and for transition functions (`digital_module`). Type `state` is a record that maintains the lists of signals applied at any time on its ports. It has one list of signals for each of the three port categories, and a port of the system is identified by its position in the list of the corresponding category. In the rest of this paper the term *signal* will sometimes be used instead of *port*, so that ‘signal x ’ means ‘the signal present at port x .’ The transition function type `digital_module` is time-dependent and has the signature $[time \rightarrow [state \rightarrow state]]$. The theory includes also a number of auxiliary functions to build lists of ports (i.e., of signals) and to select a specific port of a module, such as `ports(n)`, `ports(s, n)`, etc. The first definitions of the theory follow.

```
digital_modules_th THEORY
BEGIN
  IMPORTING signals_th
  %-- type definitions
  ports: TYPE = list[signal]
  state: TYPE = [# input: ports, output: ports,
                 internal: ports #]
  digital_module: TYPE = [time -> [state -> state]]

  %-- port constructors
  ports(n: nat): RECURSIVE
    {p: ports | length(p) = n} =
    IF n = 0 THEN null
    ELSE cons(constval(U), ports(n - 1)) ENDIF
    MEASURE n

  ports(s: signal, n: nat): RECURSIVE
    {p: ports | length(p) = n} =
    IF n = 0 THEN null
    ELSE cons(s, ports(s, n - 1)) ENDIF
    MEASURE n
```

```
%-- port selectors
port(p: ports, i: below(length(p))): signal =
  nth(p, i)

% ...
END digital_modules_th
```

Types state and digital_module are very general, and they are refined by subtyping in the theories for basic digital modules and composite digital modules, discussed below.

E. Basic Digital Modules

Basic digital modules are elements without a visible internal structure, defined only by their input and output ports and by their transition function. The state of a basic module has an empty list of internal signals, and the lists of input and output signals have a predefined length.

Basic modules are classified into two categories, *combinatorial* and *sequential*. The output signals in the next state of combinatorial modules (i.e., logic gates) depend only on the output signals of the current state, whereas in sequential modules (such as timers and flipflops) the output signals in the next state depend on both the input and the output signals of the current state.

The theory is parametric with respect to a delay parameter, representing the time needed by the component to change its outputs when its inputs change.

In addition to the parameterized definitions for the state and transition function types, the theory contains a state constructor (new_state). Part of the theory is shown below.

```
basic_digital_modules_th[delay: nonneg_real]: THEORY
BEGIN
  IMPORTING digital_modules_th

  state(nIN, nOUT: nat): TYPE =
    {s: state | length(input(s)) = nIN AND
      length(output(s)) = nOUT AND
      length(internal(s)) = 0 }

  basic_digital_module(nIN, nOUT: nat): TYPE =
    [time -> [state(nIN, nOUT) -> state(nIN, nOUT)]]

  % ...
END basic_digital_modules_th
```

This theory is imported by other theories that define various classes of basic blocks, such as logic gates, timers, and flipflops, presented in the following.

1) Logic gates: The logic_gates_th theory defines the transition functions of the basic combinatorial gates. The theory is parameterized by the propagation delay of the gates.

As the state is defined by the *signals* at the ports (and not the instantaneous values), the new state will normally be equal to the previous one, unless the environment applies different signals to the inputs (e.g., a pulse replaces a constant level). The definition for the NOR gate is shown below.

```
logic_gates_th[delay: nonneg_real]: THEORY
BEGIN
  IMPORTING basic_digital_modules_th

  gateNOR: basic_digital_module(2, 1) =
    LAMBDA (t: time): LAMBDA (s: state(2, 1)):
      s WITH [output := ports(time_shift(
        sNOR(port0(input(s)), port1(input(s))),
        delay))]
```

```
% ...
END basic_digital_modules_th
```

2) Timers: The timers_th theory defines devices that generate a single pulse when they receive a rising edge on their input port. The pulse duration is a parameter of the device. Their response to the input depends on previous values of the output and possibly of the input(s). The theory defines also resettable timers, whose output drops to zero on receiving a rising edge at the reset port. An excerpt of the PVS theory follows. More definitions in Appendix A.

```
timers_th[delay: nonneg_real]: THEORY
BEGIN
  IMPORTING basic_digital_modules_th

  %--timer
  timerM(d: posreal): basic_digital_module(1, 1) =
    LAMBDA (t: time):
      LAMBDA (s: state(1, 1)):
        IF rising_edge?(port0(input(s)), t) AND
          zero?(port0(output(s)), t)
        THEN s
        WITH [output := ports(pulse(t + delay, d))]
        ELSE s ENDIF
      % ...
END timers_th
```

3) Flip-flops: The flipflop_th theory defines 1-bit memory registers. Let us consider the SR flip-flop (Figure 1(a)). Ports *s* and *r* are the set and reset terminals, the stored bit is on the output marked *q*, and *q'* is its complement. Ports *q* and *q'* hold their previous value when *s* and *r* are both zero. If *s* becomes one while *r* is zero, then *q* is one, and stays at one even after *s* returns zero. Similarly, if *r* becomes one while *s* is zero, then *q* is zero, and stays at zero even after *r* returns zero. The PVS specification of the SR flip-flop is shown in Appendix A.

F. Composite Digital Modules

Basic digital modules can be connected together to create *composite digital modules*. The corresponding theory contains only the high-level definition for the state and the transition function, and for a state constructor (not shown).

```
state(nIN, nOUT, nINT: nat): TYPE =
  {s: state | length(input(s)) = nIN AND
    length(output(s)) = nOUT AND
    length(internal(s)) = nINT}

composite_digital_module(nIN, nOUT, nINT: nat): TYPE =
  [time -> [state(nIN, nOUT, nINT)
    -> state(nIN, nOUT, nINT)]]
```

A composite module is modelled by the composition of the transition functions of its components, whose form depends on the interconnections of the components.

In order to build the composite module, one must first define the *system* state, i.e., the union of its input, output, and internal ports. Then the subsets of the composite system state relative to the components (*component substates*) must be identified. Then a *composite transition function* is defined along the following lines: (i) Each port of the composite module is assigned a unique name by equating the port to a variable of type signal in a LET expression (e.g., *r* =

`port0(input(st))` gives the name `r` to the first input port of state `st`; (ii) for each basic component, we define its current substate by selecting its input and output signals from the current system state; (iii) for each basic component, we define its next substate as a variable of type `state`, and we equate it to the basic component's transition function applied to the current substate defined in the previous step; (iv) the output signals of the new system state are the union of the output signals of the new substates of the basic components connected to the system output; (v) the internal signals of the next system state are the union of the internal signals of the new substates of the basic components.

The composite transition function applies the transition functions of the basic components to the respective substates, obtaining a set of new substates that may not be consistent. Suppose, for example, that a composite module M is made of two inverters m_1 and m_2 in cascade, and that in the initial state there is a constant zero at the input of m_1 , a constant one between m_1 and m_2 , and a constant zero at the output of m_2 . If, at time t , the input signal to m_1 becomes a step function, the evaluation of the composite transition function places an inverted step between m_1 and m_2 , but leaves a constant zero at the output of m_2 , since its transition function is computed with the previous substate. Therefore, the final state of a transition is computed by an iterative process (similar to a fixed-point computation) that repeatedly applies the transition function until a *consistent* state is reached, i.e., a state s such that $s = f_T(s)$, where f_T is the composite transition function.

As an example, we show the composite module of the SR flip-flop built from a pair of cross-coupled NOR logic gates. With reference to Figure 1(b), in this example port `x01` is renamed as `r1`, and `x10` as `s1`.

```
flipflopSR: composite_digital_module(2, 2, 2) =
LAMBDA (t: time):
  LET st: state(2, 2, 2):
    LET r = port0(input(st)),
        s = port1(input(st)),
        q = port0(output(st)),
        q_prime = port1(output(st)),
        r1 = port0(internal(st)),
        s1 = port1(internal(st)),
        nor0 = gateNOR[tres](t)(new_state(2, 1)
          WITH [input := ports(r, r1),
                output := ports(q)]),
        nor1 = gateNOR[tres](t)(new_state(2, 1)
          WITH [input := ports(s, s1),
                output := ports(q_prime)])
    IN st WITH [output := ports(port0(output(nor0)),
                                port0(output(nor1))),
                internal := ports(port0(output(nor1)),
                                port0(output(nor0)))]
```

In the system transition function `flipflopSR`, we let signal `r` be the signal on the first input port (`port0`) of the current system state `st`, and similarly for `s`, `q`, `q_prime`, `s1`, and `r1`. Then, substate `nor0` of gate G_0 is the result of transition function `gateNOR`. The argument of this function is a state with input signals `r` and `r1`, and output signal `q`. A similar description applies to `nor1`.

The state returned by `flipflopSR` is the current state `st` with the output signals set equal to the output signals of the next substates of the two NOR gates. Similarly, the internal ports are set equal to the output signals due to the cross-coupling of NOR gates.

V. THE EVENT-DRIVEN SIMULATOR

This section describes an event-driven simulator of digital modules. First, we introduce *events*, i.e., instants when a signal may change its value. Second, we extend the specification of the system with events. Third, we present the event-driven simulation engine, which uses the extended specification to evaluate the system only at specific instants, instead of at periodic steps as in time-driven approaches [31].

A. Events

Theory `events_th` defines the type `event` as a record with fields `t`, the instant of a single event or of the first of a series of periodic events, and `T`, the period of the series (single events have $T=0$). The theory includes the ordering relation between events and operations on list of events. Some definitions are shown below.

```
BEGIN IMPORTING time_th
  event: TYPE = [# t: time, T: interval #];
  <(e1, e2: event): bool =
    (t(e1) < t(e2)) OR
    (t(e1) = t(e2) AND T(e1) < T(e2))
```

B. Annotated Signals

In theory `annotated_signals_th` we annotate the formal specification of signals with the list of events associated with each signal. We redefine the type `signal` as a record with the fields `val`, the functional specification of the signal, and `evts`, the set of instants when the waveform changes value. For example, the set of events associated with a constant level generator is empty, while the set of events associated with a pulse generator at time τ and duration d contains events τ and $\tau + d$, both with period $T = 0$.

The basic operators on signals are re-defined to calculate the events of the resulting signal, whose events are the union of events of the operator parameters.

Some events in the resulting signal may not affect the signal value. For example, in the case of `s0R`, if initially one of the inputs is a constant one, no set of events on the other input causes any change in the output. Such redundant events, however, do not affect the simulation results. The event annotation is therefore correct as the application of operators to signals yields a signal whose annotation contains all the instants when the signal changes according to its definition. An informal justification of this statement follows in the next paragraphs.

First, we consider the basic signals, having a finite number of events: `constval`, which has a given constant logic level at all instants; `step`, which has logic level *zero* at all instants before a given time t , and *one* for all other instants; `pulse`, which has logic level *one* at all instants in a given interval $[t, t+d]$, and *zero* for all other instants. Therefore, the annotated versions

TABLE I
OPERATOR EVENT ANNOTATIONS.

Operator	Events
NOT(s_1)	events(s_1)
AND(s_1, s_2)	events(s_1) \cup events(s_2)
OR(s_1, s_2)	events(s_1) \cup events(s_2)
timeshift(s_1, D)	add D to the time of each $e \in$ events(s_1)
periodic(s_1, T)	make each non-periodic $e \in$ events(s_1) periodic with period T , keep periodic ones unchanged

of *constval* has no events, *step* has one event at the instant of the rising edge of the signal, and *pulse* has two events, one at the instants of the rising edge and one at the falling edge of the signal. The definitions for these signals are as follows:

```
constval(v: logic_level): signal =
  (# val := LAMBDA (t: time): v,
   evts := new_event(0) #)

step(tau: time): signal =
  (# val := LAMBDA (t: time):
     IF t >= tau
       THEN one ELSE zero ENDIF,
     evts := new_event(tau) #)

pulse(tau: time, d: posreal): signal =
  (# val := LAMBDA (t: time):
     IF t >= tau AND t < tau + d
       THEN one ELSE zero ENDIF,
     evts := new_event(tau) + new_event(tau + d) #)
```

We now consider the signal operators, whose behaviour with respect to signal annotation is shown in Table I. As an example, the following fragment shows the definition of the SNOR operator:

```
BEGIN IMPORTING events_th, logic_levels_th
SNOR(s1a, s2a: signal): signal =
  LET s1 = val(s1a), s2 = val(s2a),
  f = LAMBDA (t: time):
    IF one?(s1(t)) OR one?(s2(t)) THEN zero
    ELSIF zero?(s1(t)) AND zero?(s2(t)) THEN one
    ELSE U ENDIF,
    e = evts(s1a) + evts(s2a)
  IN (# val := f, evts := e #)
```

Let s_i be a correctly annotated signal occurring as the i -th operand of an operator. Let s be the signal generated by the operator, and let \mathcal{E}^s be the set of events that annotate signal s . Let $\tau(e)$ be the time value (i.e., the value of the t field) of an event e , and let $\omega(e, i)$ be the time of the only occurrence of event e , if $i = 0$, or of its i -th occurrence otherwise. We finally define a non-periodic signal s as *alive in an interval* I if I is the shortest time interval containing all events of s .

The *logical operators* annotate s with the union of the events of s_i . The annotation is correct because the signal generated by any logical operators may change level only at the instants in which at least one of the signals occurring as an operand changes level.

The *timeshift operator* annotates s with a set of events \mathcal{E}^s where $\forall e' \in \mathcal{E}^s, \exists e \in \mathcal{E}^{s_1}$ such that $\tau(e') = \tau(e) + D$, where D is the offset that delays (positive offset) or advances (negative offset) the signal waveform. The annotation is correct

because a signal delayed by D has all its original events postponed of D , and a signal advanced by D has all its original events anticipated by D .

The *periodic operator* annotates s with a set of periodic events \mathcal{E}^s where $\forall e' \in \mathcal{E}^s, \forall i \in \mathbb{IN}, \exists e \in \mathcal{E}^{s_1}$ such that $\omega(e', i) = \tau(e) + iT$, where T is the period parameter of the operator. The annotation is correct because the periodic extension of a signal alive in an interval I has all events repeated with a period T , where T is greater than the length of I .

Annotated signals carry all the information needed by the simulator to handle events, so the specification of the digital modules remains unchanged.

C. Simulator

The simulator maintains a list of events (*worklist*), initialised with the starting time of the simulation. The events are listed in ascending order without duplicates. At each simulation step, the simulator extracts the first event (*current event*) from the worklist, and then it computes the next state by applying the system transition function at the time specified by the event. Then, the events associated with the signals in the generated state are inserted in the working list, provided that they are not earlier than the current event.

1) *Worklist*: Theory *worklist_th* defines the type *worklist* as a list of events, provides the function *get_first* that, given a current time, returns the first event associated with a set of signals and greater than the current time, and the function *update_wl* that updates the worklist. Function *update_wl* finds the new events in the next state and inserts them in the worklist. Note that, since the model of the system may contain ideal modules that update instantaneously their output ports, function *update_wl* must not remove the current event from the worklist as long as the generated state is not consistent (Section IV-F). For this reason, if the next state is different from the current state, then also the current event is inserted in the worklist.

The PVS specifications of these simple worklist manipulations are not shown.

2) *Simulation Engine*: The simulation engine applies the system transition function and returns the state of the system after a certain number of steps. It uses a customisable *dump* function to output a simulation trace.

The input parameters are the maximum number of steps, the system transition function, the worklist, the output stream for the trace, and the names of the signals. The function is called with an initial worklist containing all events of the initial state and an event for the initial time.

At each step, the function (i) gets the simulation time from the first event in the worklist, (ii) generates the next system state, (iii) updates the worklist, and (iv) outputs the system state.

The simulation terminates when either the new worklist is empty, or the maximum number of steps is reached. The PVS specification of the function follows.

```

simulate_system(n_steps: nat)
  (f: [time -> [state -> state]])
  (wl: worklist) (outf: OStream, pn: port_names):
RECURSIVE [state -> state] =
LAMBDA (s: state):
  IF n_steps > 0 AND length(wl) > 0
  THEN
    LET curr_t = t(get_first(wl)),
      s_prime = update_state(s)(curr_t, f),
      wl_prime = update_wl(wl)(curr_t, s, s_prime),
      dbg      = dump(outf, pn, s, s_prime,
                      wl, wl_prime, curr_t)
    IN simulate_system(n_steps - 1)(f)(wl_prime)
        (outf, pn)(s_prime)
  ELSE s ENDIF

```

The following excerpt shows how the digital module `flipflopSR` is simulated. In function `sim_flipflopSR`, the initial state is constructed from the signals at the ports, the worklist is initialised, and `simulate_system` is invoked with the transition function as an argument. The `reset` port is initially fed with a constant zero signal, the `set` port with a pulse of 4s at time 0.3, and `q` (`q'`) holds a constant zero (one).

```

sim_flipflopSR(N_STEPS: nat): bool =
LET r = constval(zero), s = pulse(0.3, 4),
  q = constval(zero), q_prime = constval(one),
  r1 = q_prime, s1 = q,
  initial_st = new_state(2, 2, 2)
  WITH [input := ports(r, s),
        output := ports(q, q_prime),
        internal := ports(r1, s1)],
  initial_wl = worklist(initial_st, 0),
  final_s = simulate_system(N_STEPS)(flipflopSR)
        (initial_wl)(outf, pn)(initial_st)
  IN TRUE

```

The simulation trace can be a list of event times, signal values and worklist contents at each step, or a *Value Change Dump* [5] output, readable by a visualisation tool.

D. Automated Execution of Test-Cases

In this section we show how PVS constructs can be conveniently used to execute test cases, relying on the PVS ground evaluator. The ground evaluator interprets a universal quantifier by generating all possible values for the quantified variable (provided it has a discrete type) and evaluating the formula for each value. Universal quantifiers may then be used much like `for` instructions in imperative programming languages.

In the following example, the `test_flipflopSR` function uses the `FORALL` quantifier to generate all possible combinations of logical levels. Each combination defines an initial state for an SR flip-flop, and each state is used to compute a next state. The ground evaluator implicitly transforms the universally quantified formula into a loop that, at each iteration, applies the transition function and prints out the values at the ports in the initial and in the next state.

```

test_flipflop_th: THEORY
BEGIN %--imports omitted
%
discrete_time: TYPE = below(2)
test_flipflopSR: bool =
  FORALL (t_set, t_reset: discrete_time):
    FORALL (v1, v2: logic_level):

```

```

      v1 /= v2 IMPLIES
      (LET initial_st = new_state(2, 2, 2)
       WITH [input := ports(pulse(t_reset, 1),
                           pulse(t_set, 1)),
             output := ports(constval(v1),
                            constval(v2)),
             internal := ports(constval(v2),
                               constval(v1))],
       initial_wl = worklist(initial_st, 0),
       final_s = simulate_system(5)(flipflopSR)
             (initial_wl)(outf,pn)(initial_st)
       IN TRUE)
      %
END test_flipflop_th

```

In Appendix B, we show an excerpt of the output generated by the above function, where each test case shows the signal values at the initial state (generated by the variable quantifiers) and the values at successive steps of the simulation.

VI. CASE STUDIES: A STEPWISE SHUTDOWN LOGIC

As an illustration of the practical applicability of the framework presented in this paper, we report on a simple case study from the field of Instrumentation and Control for NPPs. Two high-level descriptions of a control logic, expressed as Function Block Diagrams [32], have been manually translated into PVS specifications using the presented framework, and the specifications have been animated to simulate the control logic. Simulated test cases have been automatically generated, allowing a possible malfunction to be detected at this early stage of development.

A. Description of a Stepwise Shutdown Logic

A *stepwise shutdown* process keeps process variables (such as, e.g., temperature or neutron flux) within prescribed thresholds by applying a corrective action (e.g., inserting control rods) not immediately to its full extent, but gradually, in a series of discrete steps separated by settling periods.

A Stepwise Shutdown Logic (SSL) was analysed in [33] with a model checking approach. The framework proposed in this paper is used to analyse the same system.

The requirements of the SSL, as described in [33], can be informally stated as follows: if an *alarm* signal (e.g., overpressure in a pipe) is asserted, the system must assert a control signal to drive a corrective action for three seconds (*active period*), then the control signal is reset for twelve seconds (*wait period*) and the cycle is repeated until either the alarm signal is reset or a complete shutdown is reached. An operator, however, by activating a *manual trip* signal, may force the wait periods to be shortened in order to accelerate the process.

B. Design A

Figure 2 shows the main part of design A, where `m` is the manual trip, `p` is an alarm signal, and `out` is the control signal. When all signals are low, the output `t2_out` of timer T2 is low, and the AND gate is enabled. When `p` is asserted, its rising edge passes through the AND gate to the input of the T1 timer that sends a 3 s pulse to the output. The output is fed back to the input of T2, a resettable timer with a pulse duration

```

systemA: composite_digital_module(nIN, nOUT, nINT) =
LAMBDA (t: time):
  LAMBDA (st: state(nIN, nOUT, nINT)):
    LET m = port0(input(st)), p = port1(input(st)), out = port0(output(st)),
      t2_in = port0(internal(st)), t2_out = port1(internal(st)),
      %... similar definitions for orl_in, and_en, and_out
      rtimer = rtimerM[T1](D2)(t)(new_state(2,1))
      WITH [input:=ports(t2_in,m), output:=ports(t2_out)],
      orl = gateOR[T0](t)(new_state(2,1)) WITH [input:=ports(orl_in,p), output:=ports(orl_out)],
      inh_and = gateANDH[T0](t)(new_state(2,1)) WITH [input:=ports(and_en, and_in), output:=ports(and_out)],
      timer = timerM[T2](D1)(t)(new_state(2,1)) WITH [input:=ports(t1_in), output:=ports(out)]
    IN st WITH [input := ports(m, p),
      output := ports(port0(output(timer))),,
      internal := ports(port0(output(timer)),, port0(output(rtimer)), m,
                        port0(output(orl)),, port0(output(rtimer)), port0(output(orl)),,
                        port0(output(inh_and)),, port0(output(inh_and)))]
  
```

Fig. 3. PVS model of the Stepwise Shutdown Logic.

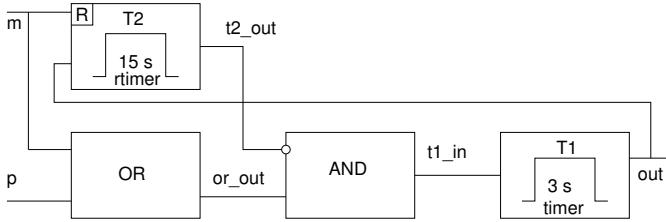


Fig. 2. A simplified view of a stepwise shutdown logic, design A.

of 15 s. The output pulse of T2 disables the AND gate that in turn resets the input of T1. Since T1 is not resettable, its output pulse lasts for three seconds, then returns to low for the remaining 12 s of the T2 pulse. After this wait period, the output of T2 goes low, the AND gate is enabled, and T1 starts a new pulse if an input signal is still asserted.

If p is high, and m is asserted during a wait period, T2 is reset and its output enables the AND gate, allowing the trip signal to reach T1 and restart it at the end of the 3 s pulse.

The SSL is modelled by the `systemA` transition function (see Figure 3), according to the guidelines in Section IV. All components are assumed to introduce a delay of 1 ms.

In the rest of this section we show some simulated situations. First we examine a few scenarios generated with the procedure described in Section V-D.

1) Automated Execution of Test-Cases: Assuming that an overpressure (say) signal p is asserted at time $t = 1$ s and remains constant thereafter, we study the possible effects of a manual trip request by letting the time of occurrence of the request vary over a given interval. More precisely, we model the request as a 1 s pulse on the m line, with an initial instant t_0 varying between 1 and N seconds, with steps of one second. This is done by the following code:

```

sim_systemA_test(N:nat): bool =
FORALL(t0: below(N)):
  LET
    initial_st = new_state(nIN, nOUT, nINT)
    WITH [input := ports(pulse(t0,1), step(1)),
          output := ports(constval(zero)),
          internal := ports(constval(zero), nINT)],
    initial_wl = worklist(initial_st, 0),
  
```

```

final_s = simulate_system(NSTEPS) (systemA)
  (initial_wl) (outf, pn) (initial_st)
IN TRUE
  
```

The maximum number of simulation steps for each run (NSTEPS) was set at one hundred.

The simulator outputs of the initial four test cases ($t \in \{1, 2, 3, 4\}$) are summarised in Tables II, III, IV, and V.

Examining the data recorded in these tables, we notice that two different behaviours are exhibited by the system, as shown by Tables II and V on one hand, and Tables III and IV on the other hand. With the manual trip signal activated at $t_0 = 1$ s (Table II) and at $t_0 = 4$ s (Table V), the simulation executes the maximum requested number of steps, up to a simulated time of 271.039 s and 259.037 s, respectively.

TABLE II
AUTOMATED TESTS FOR DESIGN A ($t_0 = 1$).

time	m	p	out	worklist
0.	0	0	0	[0 001 1 2]
0.001	0	0	0	[0.002 1 2]
0.002	0	0	0	[1 2]
1.	1	1	0	[1 001 2]
1.001	1	1	0	[1.002 2]
1.002	1	1	1	[1.003 2]
1.003	1	1	1	[1.004 2]
1.004	1	1	1	[2]
2.	0	1	1	[2 001]
...
256.035	0	1	0	[256.036]
256.036	0	1	1	[256.037]
256.037	0	1	1	[256.038]
256.038	0	1	1	[259.036]
259.036	0	1	0	[271.037]
271.037	0	1	0	[271.038]
271.038	0	1	1	[271.039]
271.039	0	1	1	[271.04]

We observe that the worklist is not empty at the last step, and deduce that the simulation could proceed for a greater (possibly unbounded) number of steps. This is supported by the periodic pattern shown by the values of the out signal. This is the expected behaviour, where the output signal skips a wait period when the manual trip button is depressed and, after

TABLE III
AUTOMATED TESTS FOR DESIGN A ($t_0 = 2$).

$t_0 = 2$				
time	m	p	out	worklist
0.	0	0	0	[0.001 1 2 3]
0.001	0	0	0	[0.002 1 2 3]
0.002	0	0	0	[1 2 3]
1.	0	1	0	[1.001 2 3]
1.001	0	1	0	[1.002 2 3]
1.002	0	1	1	[1.003 2 3]
1.003	0	1	1	[1.004 2 3]
1.004	0	1	1	[2 3]
2.	1	1	1	[2.001 3]
2.001	1	1	1	[2.002 3]
2.002	1	1	1	[3]
3.	0	1	1	[3.001]
3.001	0	1	1	[3.002]
3.002	0	1	1	[4.002]
4.002	0	1	0	[]

TABLE IV
AUTOMATED TESTS FOR DESIGN A ($t_0 = 3$).

$t_0 = 3$				
time	m	p	out	worklist
0.	0	0	0	[0.001 1 3 4]
0.001	0	0	0	[0.002 1 3 4]
0.002	0	0	0	[1 3 4]
1.	0	1	0	[1.001 3 4]
1.001	0	1	0	[1.002 3 4]
1.002	0	1	1	[1.003 3 4]
1.003	0	1	1	[1.004 3 4]
1.004	0	1	1	[3 4]
3.	1	1	1	[3.001 4]
3.001	1	1	1	[3.002 4]
3.002	1	1	1	[4]
4.	0	1	1	[4.001]
4.001	0	1	1	[4.002]
4.002	0	1	0	[]

the button is released, produces a series of regularly spaced pulses as long as the overpressure signal is active.

The other two tables (Tables III and IV), instead, show that the simulation stopped early, with an empty worklist at the last step. This proves that the system ‘freezes’ with the output stuck at zero, whereas it should produce periodic pulses.

Comparing these cases, and other cases not shown, we may formulate the hypothesis that the logic malfunctions when a manual trip is issued during the active period of the output pulse, as in the situations illustrated by Tables III and IV. To test this hypothesis, we explore some hand-crafted scenarios, discussed in the rest of the section.

2) *No manual trip:* Signal p is a step function with the rising edge at $t = 1$ s, and signal m is a constant zero (no manual intervention). The control logic produces a series of pulses that drive the plant towards a shutdown, as expected (Figure 4).

3) *Manual trip in the wait period:* Signal p is a step function with the rising edge at $t = 1$ s and signal m is a step function with the rising edge at $t_0 = 5$ s. This means

TABLE V
AUTOMATED TESTS FOR DESIGN A ($t_0 = 4$).

$t_0 = 4$				
time	m	p	out	worklist
0.	0	0	0	[0.001 1 4 5]
0.001	0	0	0	[0.002 1 4 5]
0.002	0	0	0	[1 4 5]
1.	0	1	0	[1.001 4 5]
1.001	0	1	0	[1.002 4 5]
1.002	0	1	1	[1.003 4 5]
1.003	0	1	1	[1.004 4 5]
1.004	0	1	1	[4 5]
4.	1	1	1	[4.001 5]
...
244.037	0	1	1	[247.035]
247.035	0	1	0	[259.036]
259.036	0	1	0	[259.037]
259.037	0	1	1	[259.038]

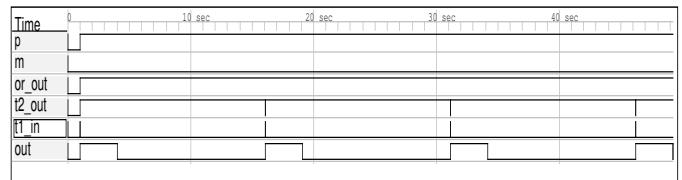


Fig. 4. Simulation of design A, no manual trip.

that the trip switch is pushed during the first wait period. As expected, that wait period is interrupted, a new 3 s output pulse is generated, and the subsequent pulses are generated with the normal 15 s cycle, since the trip switch has not been released and the resettable timer responds only to a rising edge (Figure 5).

4) *Manual trip in the active period:* In this instance, signal p is a step function with the rising edge at $t = 1$ s and signal m is a pulse of duration 1 s starting at $t_0 = 2$ s, followed by another pulse of duration 3 s at $t_1 = 10$ s. In this case, the manual intervention occurs during the active period of the first output pulse. Contrary to the requirements, after the end of this output pulse, the output is stuck at zero and no further corrective action takes place, even if the alarm (high pressure) persists and the manual trip switch is pressed again. A fundamental safety requirement is thus violated (Figure 6). The PVS code for this critical case follows.

```
sim_system3A: bool =
LET initial_st =
  new_state(nIN, nOUT, nINT)
  WITH [input := ports(pulse(2,1)+pulse(10,3),
                      step(1)),
        output := ports(constval(zero)),
        internal := ports(constval(zero), nINT)],
       initial_wl = worklist(initial_st, 0),
       final_s = simulate_system(NSTEPS) (systemA)
                           (initial_wl) (outf, pn) (initial_st)
  IN TRUE
```

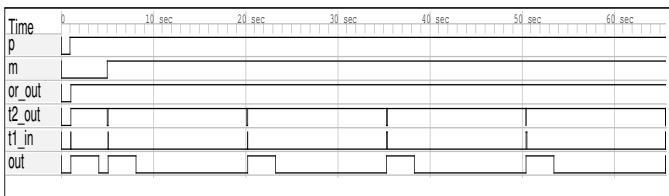


Fig. 5. Simulation of design A, manual trip in the wait period.

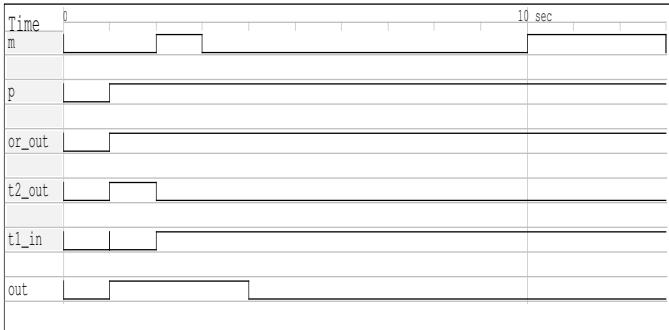


Fig. 6. Simulation of design A, manual trip in the active period.

C. Design B

Figure 7 shows the main part of design B. This design is identical to design A except for the management of the manual trip: in design B, the manual trip signal is fed to a 3 s timer whose output is ORed with the overpressure signal and with the output of the other 3 s timer, and the 15 s timer is not resettable. We translated this schema into a PVS specification and simulated it under the same scenarios used for checking design A. The simulation experiments did not detect any malfunctions. In particular, Figure 8 shows the output for the problematic scenario of Section VI-B4 that caused a safety violation for design A. With design B, the logic honours the two manual interventions, then it keeps issuing 3 s pulses, thus fulfilling the safety requirement.

The safety of Design B in the situation considered can be proved along the lines of the following proof sketch.

- 1) As explained in Section IV-F, the transition function of the compound module must be applied repeatedly until the actual successor state is computed. We define a *micro step* as a single application of the transition function, and a *macro step* as a sequence of micro steps leading from a consistent state to its consistent successor.
- 2) Starting from an initial state where signal m is a constant zero, p is a step function at $t = 1$, and all internal signals are assumed to be constant zeroes, we prove the existence of a sequence of micro steps leading to a consistent state at $t = 1$. This part of the proof is a set of simple lemmas, one for each n -th micro step, of the form:

$$\text{init} = s(n-1) \wedge \text{next} = f(t)(\text{init}) \Rightarrow \text{next} = s(n)$$

where f is the transition function, and $s(n-1)$ and

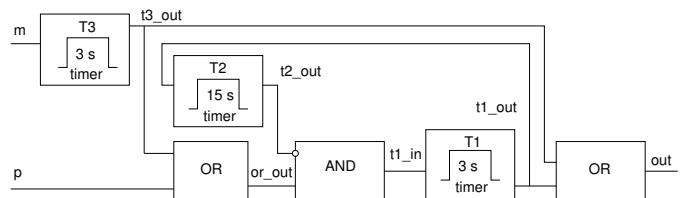


Fig. 7. A simplified view of a stepwise shutdown logic, design B.

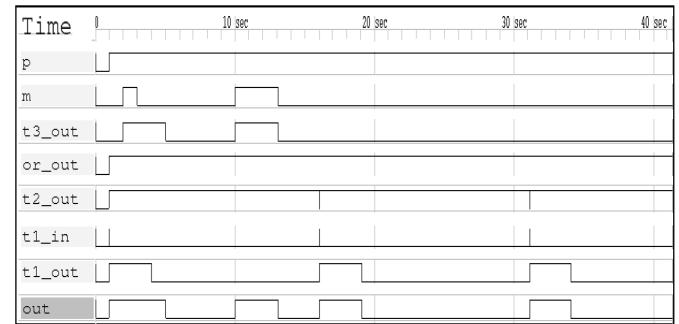


Fig. 8. Simulation of design B, manual trip in the active period.

$s(n)$, i.e., the initial and final states of the micro step, are given.

- 3) Then we prove, in the same way, that consistent states are reached at time $t = 4$ (after the 3 s pulse) and $t = 16$ (after the 15 s pulse).
- 4) We prove that the state at $t = 16$ is equivalent (modulo a 15 s translation) to the state at $t = 1$. Therefore, with the given signals at the input (in particular, without manual trip), the system has a periodic behavior, as it returns to the same conditions every 15 s. More precisely, it produces 3 s pulses (generated by timer T1, see Figure 7) with a 15 s period.
- 5) Since signal m is fed to a 3 s timer whose output is ORed with the output of T1, this signal cannot suppress the control signal, therefore Design B is safe for any possible timing of signal m .

A small sample of the lemmas used in the proof are in Appendix C.

VII. CONCLUSION AND FUTURE WORK

In the present work, a framework for the simulation of control logics specified in the higher-order logic of the PVS has been introduced. The framework is based on a library of purely logic specifications for typical control system components, and an approach to define an event-driven simulator capable of executing the logic specifications is shown. The library includes theories to model logic signals over time, where time is a variable in the domain of real numbers. The simulator is based on the paradigm of event-driven-simulation, and its core component is defined as a function in the higher-order logic language of the PVS theorem proving environment. The approach has been applied to a simple case study in the field of nuclear power plants. The same case study had been

previously studied by other researchers with a model checking approach [33]. Further, compliance of one of the designs with a safety requirement has been demonstrated by theorem proving.

This work is part of our current research activity aiming at developing a simulation and analysis framework for control logics that enables developers to rely both on simulation and theorem proving to assess the correctness of specifications and designs.

REFERENCES

- [1] C. Bernardeschi, L. Cassano, A. Domenici, and P. Masci, "Debugging PVS specifications of control logics via event-driven simulation," in *First International Conference on Computational Logics, Algebras, Programming, Tools, and Benchmarking (COMPUTATION TOOLS 2010)*. Lisbon, Portugal: IARIA, November 21–26 2010.
- [2] S. Owre, J. Rushby, N. Shankar, and F. von Henke, "Formal Verification for Fault-Tolerant Architectures: Prolegomena to the Design of PVS," *IEEE Trans. on Software Engineering*, vol. 21, no. 2, pp. 107–125, 1995.
- [3] "Railway applications – Software for railway control and protection systems," CENELEC, European Committee for Electrotechnical Standardization, Tech. Rep. EN 50128:2001 E, 2001, european standard.
- [4] "Software for Computer Based Systems Important to Safety in Nuclear Power plants," IAEA, International Atomic Energy Agency, Tech. Rep. NS-G-1.1, 2000.
- [5] "IEEE Standard Verilog Hardware Description Language," IEEE, Tech. Rep. IEEE Std 1076-2000, 2000.
- [6] "IEEE Standard VHDL Language Reference Manual," IEEE, Tech. Rep. IEEE Std 1076-2000, 2000.
- [7] E. Clarke, O. Grumberg, and D. Peled, *Model checking*. MIT Press, 1999.
- [8] E. M. Clarke, E. A. Emerson, and A. P. Sistla, "Automatic verification of finite-state concurrent systems using temporal logic specifications," *ACM Transactions on Programming Languages and Systems*, vol. 8, pp. 244–263, 1986.
- [9] J. McCarthy, "Checking mathematical proofs by computer," in *Symposium on Recursive Function Theory*. American Mathematical Society, 1961.
- [10] B. J. Krämer and N. Völker, "A highly dependable computing architecture for safety-critical control applications," *Real-Time Syst.*, vol. 13, pp. 237–251, November 1997.
- [11] H. Wan, G. Chen, X. Song, and M. Gu, "Formalisation and verification of programmable logic controllers timers in Coq," *Software, IET*, vol. 5, no. 1, pp. 32–42, February 2011.
- [12] E. Jee, S. Jeon, S. Cha, K. Koh, J. Yoo, G. Park, and P. Seong, "FBDVerifier: Interactive and visual analysis of counter-example in formal verification of function block diagram," *Journal of Research and Practice in Information Technology*, vol. 42, no. 3, pp. 171–189, 2010.
- [13] V. Vyatkin and H.-M. Hanisch, "Modelling of IEC 61499 function blocks a clue to their verification," in *XI Workshop on Supervising and Diagnostics of Machining Systems*, no. 35, 2000, pp. 59–68.
- [14] D. Missal, M. Hirsch, and H.-M. Hanisch, "Hierarchical distributed controllers - design and verification," in *IEEE Conference on Emerging Technologies and Factory Automation (ETFA 2007)*, Sept. 2007, pp. 657–664.
- [15] D. Deharbe, S. Shankar, and E. Clarke, "Formal verification of VHDL: the model checker CV," in *XI Brazilian Symposium on Integrated Circuit Design*, 1998, pp. 95 –98.
- [16] D. Russinoff, "A Formalization of a Subset of VHDL in the Boyer-Moore Logic," *Formal Methods in System Design*, vol. 7, no. 1/2, pp. 7–26, 1994.
- [17] R. Boyer and J. Moore, *A Computational Logic Handbook*. Academic Press, 1988.
- [18] H. Jain, D. Kroening, N. Sharygina, and E. Clarke, "Word-level predicate-abstraction and refinement techniques for verifying RTL Verilog," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 2, pp. 366 –379, feb. 2008.
- [19] H. Jain, N. Sharygina, and E. Clarke, "VCEGAR: Verilog counterexample guided abstraction refinement," in *Tools and Algorithms for the Construction and Analysis of Systems (TACAS07)*, 2007.
- [20] S. Owre, J. Rushby, N. Shankar, and D. Stringer-Calvert, "PVS: an experience report," in *Applied Formal Methods*, ser. LNCS. Springer-Verlag, 1998, no. 531, pp. 338–345.
- [21] S. Owre, J. Rushby, N. Shankar, and M. Srivas, "A tutorial on using PVS for hardware verification," in *Theorem Provers in Circuit Design (TPCD '94)*, ser. LNCS, R. Kumar and T. Kropf, Eds. Springer-Verlag, 1997, no. 901, pp. 258–279.
- [22] M. Srivas, H. Rueß, and D. Cyrluk, "Hardware verification using PVS," in *Formal Hardware Verification: Methods and Systems in Comparison*, ser. LNCS, T. Kropf, Ed. Springer-Verlag, 1997, no. 1287, pp. 156–205.
- [23] C. Berg, C. Jacobi, and D. Kroening, "Formal verification of a basic circuits library," in *Proc. of IASTED Int. Conf. on Applied Informatics, Innsbruck (AI 2001)*. ACTA Press, 2001.
- [24] H. Pfeifer, "Formal verification of the TTP group membership algorithm," in *Formal Methods for Distributed System Development Proceedings of FORTE XIII/PSTV XX 2000*, T. Bolognesi and D. Latella, Eds. Pisa, Italy: Kluwer Academic Publishers, October 2000, pp. 3–18.
- [25] C. Bernardeschi, P. Masci, and H. Pfeifer, "Analysis of wireless sensor network protocols in dynamic scenarios," in *11th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS09)*, ser. Lecture Notes in Computer Science, vol. 5873. Springer, 2009, pp. 105–119.
- [26] P. Masci, P. Curzon, A. Blandford, and D. Furniss, "Modelling distributed cognition systems in PVS," in *4th Intl. Workshop on Formal Methods for Interactive Systems (FMIS2011)*, 2011.
- [27] R. R. Lutz, "Analyzing software requirements errors in safety-critical, embedded systems," in *Proceedings of the IEEE International Symposium on Requirements Engineering*, 1993, pp. 126–133.
- [28] S. Owre, S. Rajan, J. Rushby, N. Shankar, and M. Srivas, "PVS: combining specification, proof checking, and model checking," in *Computer-Aided Verification, CAV '96*, ser. LNCS, R. Alur and T. Henzinger, Eds. Springer-Verlag, 1996, no. 1102, pp. 411–414.
- [29] J. Crow, S. Owre, J. Rushby, N. Shankar, and D. Stringer-Calvert, "Evaluating, testing, and animating PVS specifications," Computer Science Laboratory, SRI International, Tech. Rep., 2001.
- [30] C. Muñoz, "Rapid prototyping in PVS," National Institute of Aerospace, Hampton, VA, USA, Tech. Rep. NIA 2003-03, NASA/CR-2003-212418, 2003.
- [31] A. M. Law and D. Kelton, *Simulation Modeling and Analysis*. McGraw-Hill, 2000.
- [32] "Programmable controllers - Part 3: Programming languages, ed2.0," IEC, International Electrotechnical Commission, Tech. Rep. IEC 61131-3, 2003.
- [33] K. Björkman, J. Frits, J. Valkonen, J. Lahtinen, K. Heljanko, I. Niemelä, and J. J. Hämäläinen, "Verification of Safety Logic Designs by Model Checking," in *Sixth American Nuclear Society International Topical Meeting on Nuclear Plant Instrumentation, Control, and Human-Machine Interface Technologies (NPIC&HMIT 2009)*. Knoxville, Tennessee, USA: American Nuclear Society, LaGrange Park, IL, USA, 2009, on CD-ROM.

APPENDIX A SAMPLE PVS DEFINITIONS

In this appendix we show more extensive samples from the PVS theories discussed in this paper.

A. Logic Levels

```
logic_levels_th: THEORY
BEGIN
  %-- logic level (type definition)
  logic_level: TYPE = below(4)

  %-- names of logic levels
  zero: logic_level = 0
  one: logic_level = 1
  z: logic_level = 2    %-- high impedance
  U: logic_level = 3    %-- unknown

  %-- logical AND in a four-valued logic
  lAND(v1, v2: logic_level): logic_level =
    IF one?(v1) AND one?(v2) THEN one
    ELSIF zero?(v1) OR zero?(v2) THEN zero
    ELSE below(4)
```

```

ELSE U ENDIF
%-- logical OR in a four-valued logic
lOR(v1, v2: Logic_level): Logic_level =
  IF one?(v1) OR one?(v2) THEN one
  ELSIF zero?(v1) AND zero?(v2) THEN zero
  ELSE U ENDIF

%-- logical NOT in a four-valued logic
lNOT(v: Logic_level): Logic_level =
  IF one?(v) THEN zero
  ELSIF zero?(v) THEN one
  ELSE U ENDIF

% ...
END logic_levels_th

```

B. Signals

```

signals_th: THEORY
BEGIN
  IMPORTING time_th, logic_levels_th

  %-- signal (type definition)
  signal: TYPE = [time -> logic_level]

  %-- symbolic constant of the minimum time
  % between two observable variations in a signal
  tres: posreal

  %-- definition of basic waveforms
  constval(v: logic_level): signal =
    LAMBDA (t: time): v

  step(tau: time): signal =
    LAMBDA (t: time):
      IF t >= tau THEN one ELSE zero ENDIF

  pulse(tau: time, d: posreal): signal =
    LAMBDA (t: time):
      IF t >= tau AND t < tau + d
        THEN one
        ELSE zero
      ENDIF

  %-- periodic signal constructor
  periodic(s: signal, T: interval): signal =
    LAMBDA (t: time):
      LET tmod =
        IF T > 0
          THEN t - T * floor(t / T)
        ELSE t
      ENDIF
      IN s(tmod)

  %-- time shift of the signal
  time_shift(s: signal, offset: time): signal =
    LAMBDA (t: time): s(t - offset)

  %-- logical operators in a four-valued logic
  sAND(s1, s2: signal): signal =
    LAMBDA (t: time):
      IF one?(s1(t)) AND one?(s2(t))
        THEN one
      ELSIF zero?(s1(t)) OR zero?(s2(t))
        THEN zero
      ELSE U ENDIF

  sOR(s1, s2: signal): signal =
    LAMBDA (t: time):
      IF one?(s1(t)) OR one?(s2(t))
        THEN one
      ELSIF zero?(s1(t)) AND zero?(s2(t))
        THEN zero
      ELSE U ENDIF

  sNOT(s: signal): signal =
    LAMBDA (t: time):
      IF one?(s(t)) THEN zero

```

```

      ELSIF zero?(s(t)) THEN one
      ELSE U ENDIF

      % ...
END signals_th

```

The function for building periodic signals needs some explanation. The function has two arguments —the specification of a signal s in a base interval $[0, T]$, and the duration of the interval T — and generates a periodic signal by using modulo arithmetic on time instants, i.e., given a time instant t , the signal value at t is obtained by evaluating the signal at $t - T \times \lfloor t/T \rfloor$.

C. Basic Digital Modules

```

logic_gates_th[delay: nonneg_real]: THEORY
BEGIN IMPORTING basic_digital_modules_th

gateNOR: basic_digital_module(2, 1) =
  LAMBDA (t: time): LAMBDA (s: state(2, 1)):
    s WITH [output := ports(time_shift(
      sNOR(port0(input(s)), port1(input(s))), delay))]

  % ...
END basic_digital_modules_th

timers_th[delay: nonneg_real]: THEORY
BEGIN
  IMPORTING basic_digital_modules_th

  %--timer
  timerM(d: posreal): basic_digital_module(1, 1) =
    LAMBDA (t: time):
      LAMBDA (s: state(1, 1)):
        IF rising_edge?(port0(input(s)), t) AND
          zero?(port0(output(s)), t)
        THEN s
        WITH [output := ports(pulse(t + delay, d))]
        ELSE s ENDIF

  %--resettable timer (reset is input port1)
  rtimerM(d: posreal): basic_digital_module(2, 1) =
    LAMBDA (t: time):
      LAMBDA (s: state(2, 1)):
        IF rising_edge?(port1(input(s)), t)
        THEN
          IF one?(port0(output(s)), t)
          THEN s WITH
            [output := ports(sNOT(step(t + delay)))]
          ELSE s
          ENDIF
        ELSIF rising_edge?(port0(input(s)), t)
        THEN
          IF zero?(port0(output(s)), t)
          THEN s WITH
            [output := ports(pulse(t + delay, d))]
          ELSE s
          ENDIF
        ELSE s ENDIF

        % ...
END timers_th

flipflopSR: basic_digital_module(2, 2) =
  LAMBDA (t: time):
    LAMBDA (st: state(2, 2)):
      LET r = port0(input(st)),
        s = port1(input(st)),
        q = port0(output(st)),
        q_prime = port1(output(st))
      IN IF zero?(s, t) AND zero?(r, t) THEN st
      ELSIF one?(s, t) AND zero?(r, t)
        THEN IF zero?(q, t) AND one?(q_prime, t)
          THEN st WITH [output := ports

```

```

<PVSio> test_flipflopSR;
TEST 0001
<r:1, s:1, q:0, q':1, r1:1, s1:0> WL:[0 1]
t=0
<r:1, s:1, q:0, q':1, r1:1, s1:0> WL:[0.001 1]
t=0.001
<r:1, s:1, q:0, q':0, r1:0, s1:0> WL:[1 1.001]
t=1
<r:0, s:0, q:0, q':0, r1:0, s1:0> WL:[1.001]
t=1.001
<r:0, s:0, q:1, q':1, r1:1, s1:1> WL:[1.002]
t=1.002
<r:0, s:0, q:0, q':0, r1:0, s1:0> WL:[1.003]

TEST 0010
<r:1, s:1, q:1, q':0, r1:0, s1:1> WL:[0 1]
t=0
<r:1, s:1, q:1, q':0, r1:0, s1:1> WL:[0.001 1]
t=0.001
<r:1, s:1, q:0, q':0, r1:0, s1:0> WL:[1 1.001]
t=1
<r:0, s:0, q:0, q':0, r1:0, s1:0> WL:[1.001]
t=1.001
<r:0, s:0, q:1, q':1, r1:1, s1:1> WL:[1.002]
...

```

Fig. 9. Test output for SR flip-flop

```

        (step(t+delay), sNOT(step(t+delay))) ]
ELSIF zero?(s, t) AND one?(r, t)
THEN IF one?(q, t) AND zero?(q_prime, t)
THEN st WITH [output := ports
              (sNOT(step(t+delay)), step(t+delay)) ]
ELSE st ENDIF
ELSE st WITH [output := ports(2)]
ENDIF

```

APPENDIX B THE EVENT-DRIVEN SIMULATOR

This appendix contains supplementary material on the event-driven simulator.

A. Automated Execution of Test-Cases

In Figure 9 we show an excerpt of the output generated by function `test_flipflopSR` V-D. As an example, test TEST 0001 represents the case when there is a pulse on `set` and `reset` at time 0, and therefore the value of `r` and `s` is 1 in the initial state; `q` and `q_prime` have values 0 and 1. In this example, each test simulates up to five events of the system, and it can be noticed that they are not sufficient to reach a final system state (the final worklist is not empty).

APPENDIX C PROOF SKETCH

As reported in Section VI-C, the proof of the safety requirement for Design B relies on a sequence of lemmas, each defining the state resulting from a micro-step, i.e., a single application of the composite transition function.

As an example, the following PVS code is the lemma for the first micro-step, with the transition function computed at time $t = 1$:

```

sys_B_lemma1: LEMMA
FORALL (init, nxt: state(nIN, nOUT, nINT)):
  init =

```

```

  (# input := ports(constval(zero), step(1)),
   output := ports(constval(zero)),
   internal := rep_ports(constval(zero), nINT) #)
AND nxt = systemB(1)(init)
=>
  port0(internal(nxt)) =
    constval(zero)      % t2_in
AND port1(internal(nxt)) =
  constval(zero)      % t2_out
AND port2(internal(nxt)) =
  constval(zero)      % or1_in_1
AND port3(internal(nxt)) =
  step(1)             % or1_in_2
AND port4(internal(nxt)) =
  step(1)             % or1_out
AND port5(internal(nxt)) =
  constval(zero)      % and_en
AND port6(internal(nxt)) =
  step(1)             % and_in
AND port7(internal(nxt)) =
  constval(zero)      % and_out
AND port8(internal(nxt)) =
  constval(zero)      % t1_in
AND port9(internal(nxt)) =
  constval(zero)      % t1_out
AND port10(internal(nxt)) =
  constval(zero)      % t3_out
AND port0(output(nxt)) =
  constval(zero)      % or2_out

```

After a few micro-steps, a consistent state is reached. A state is proved to be consistent by a lemma such as the following:

```

sys_B_lemma6: LEMMA
FORALL (init, nxt: state(nIN, nOUT, nINT)):
  init =
  (# input :=
   ports(constval(zero), step(1)),
   output := ports(pulse(1, 3)),
   internal :=
     cons(pulse(1, 3),          % (0) t2_in
           cons(pulse(1, 15),    % (1) t2_out
                 cons(constval(zero), % (2) or1_in_1
                       cons(step(1),    % (3) or1_in_2
                             cons(step(1),    % (4) or1_out
                                   cons(pulse(1, 15), % (5) and_en
                                         cons(step(1),    % (6) and_in
                                               cons(spike(1),    % (7) and_out
                                                 cons(spike(1),    % (8) t1_in
                                                   cons(pulse(1, 3), % (9) t1_out
                                                     cons(constval(zero), % (10) t3_out
                                                       null)))))))))))
  #)
AND nxt = systemB(1)(init)
=>
nxt = init

```

Each lemma is proved by asserting a few simple axioms on the properties of signals and basic gates, then using the automatic PVS proof strategies `assert` and `grind`, thus requiring minimal human effort.

The CloudMIG Approach: Model-Based Migration of Software Systems to Cloud-Optimized Applications

Sören Frey and Wilhelm Hasselbring
*Software Engineering Group
 University of Kiel
 24118 Kiel, Germany
 {sfr, wha}@informatik.uni-kiel.de*

Abstract—Cloud computing provides means for reducing over- and under-provisioning through enabling a highly flexible resource allocation. Running an existing software system on a cloud computing basis can involve extensive reengineering activities during the migration. To reduce the correspondent effort, it is often possible to deploy an existing system widely unmodified in IaaS VM instances. However, this simplistic migration approach does not solve the challenge of over- and under-provisioning or scalability issues per se, as our experiments using Eucalyptus and the popular open source system Apache OFBiz show. Moreover, current migration approaches suffer from several further shortcomings. For example, they are often limited to specific cloud environments or do not provide automated support for the alignment with a cloud environment. We present our model-based approach CloudMIG which addresses these shortcomings. It aims at supporting SaaS providers to semi-automatically migrate existing enterprise software systems to scalable and resource-efficient PaaS and IaaS-based applications. To facilitate reasoning about the suitability of certain cloud environments for a given system and the degree of alignment during the reengineering process, we introduce the Cloud Suitability and Alignment (CSA) hierarchy. For example, Apache OFBiz used in our experiments is initially categorized “cloud compatible” but not “cloud optimized” as it does not exploit the cloud’s advantages.

Keywords-Approach *CloudMIG, Cloud Computing, Model-based software migration to cloud-based applications, Resource-efficient cloud-based applications, Eucalyptus, CSA hierarchy.*

I. INTRODUCTION

Most enterprise applications’ workload underlies substantial variations over time. For example, user behavior tends to be daytime-dependent or media coverage can lead to rapidly increasing popularity of provided services. These variations often result in over- or under-provisioning of data center resources (e.g., #CPUs or storage capacity). Cloud computing provides means for reducing over- and under-provisioning through supplying elastic services. Thereby, the conformance with contractually agreed Service Level Agreements (SLAs) has to be ensured. Considering legacy software systems, is there a way established enterprise applications can benefit from present cloud computing technologies? For reasoning about this issue, it is useful to clarify the main participants in providing and consuming cloud computing services. Three different roles can be

distinguished. Software as a Service (SaaS) providers (cloud users) offer software services, which are being utilized by SaaS users. For this purpose, the SaaS providers may build upon services offered by cloud providers (cloud vendors). In the following, we will employ the terms SaaS user, SaaS provider, and cloud provider.

Newly developed enterprise software may easily be designed for utilizing cloud computing technologies in a green-field project. Though, SaaS providers may also consider to grant responsibility of operation and maintenance tasks to a cloud provider for an already existing software system. Running established enterprise software on a cloud computing basis may involve extensive reengineering activities during the migration. Nevertheless, instead of recreating the functionalities of an established software system from scratch for being compatible with a selected cloud provider’s environment, a migration enables the SaaS provider to reuse substantial parts of a system. The number of system parts which might be migrated is dependent on the weighting of several parameters in a specific migration project. For example, implications concerning the performance or structural quality metrics regarding the resulting software architecture can be taken into account. Furthermore, aligning a software system to a cloud environment’s special properties during the migration process has the potential to increase the software system’s efficiency. For example, a reengineer could decide to prefer utilization of certain resources according to their pricing. Considering such kinds of favorable resource utilization and a cloud environment’s specific scalability mechanisms can improve overall resource efficiency (e.g., according to the aforementioned prioritization) and scalability. However, there are several major obstacles which can impede such migration projects. Current approaches are often limited to specific cloud environments or do not provide automated support for the alignment with a cloud environment, for instance. In this work, we propose our model-based approach CloudMIG, which addresses these shortcomings and focuses on the SaaS provider perspective. The semi-automated approach aims at assisting reengineers in migrating existing enterprise software systems to scalable and resource-efficient Platform as a Service (PaaS) and Infrastructure as a Service

(IaaS) based applications. This paper is an updated and extended version of [1]. It mainly adds two contributions to the original version. First, experiments were conducted utilizing the IaaS cloud environment Eucalyptus [2] and the open source system Apache OFBiz [3] that illustrate the limitations of simplistic migration strategies and advocate profound evaluation and reengineering measures during a migration. Second, we introduce the Cloud Suitability and Alignment (CSA) hierarchy that enables a classification of existing software systems regarding their suitability for specific cloud environments and their level of alignment after initial migration steps.

The remainder of the paper is structured as follows: The related work is described in Section II. Section III presents the experiments utilizing Eucalyptus and Apache OFBiz. These constitute an example scenario for demonstrating the shortcomings of the prevalent simplistic migration approaches that are described in Section IV. The CSA hierarchy forms a basis for reasoning about migration alternatives and is introduced in Section V. Our approach CloudMIG is then presented in the following Section VI, before Section VII draws the conclusions and outlines future work.

II. RELATED WORK

CloudMIG supports reengineers to migrate existing enterprise software systems to the cloud and to reduce complexity aligning their system with the targeted cloud environment. The general complexity of legacy system migration as well as potential measures to cope with the complexity are described in [4]. The authors in [5] sketch a research agenda in cloud technologies and summarize the currently published cloud computing literature. Issues and challenges regarding the cloud computing technology are investigated in [6]. Here, the integration of existing legacy systems in the cloud is regarded a challenging subject, as currently the cloud landscape is diversified and there is a lack of common practices and general interoperability. With CloudMIG we address the prevalent heterogeneity concerning cloud environments and strive towards a more generic migration approach.

A case study of migrating an enterprise IT system to an IaaS cloud environment is presented in [7]. The case study shows achievable costs savings in the cloud environment. However, the authors recommend to consider overall organizational implications as well. A large science database was migrated to the cloud in [8]. The main hurdles the authors faced lay in the transfer of huge amounts of data and performance degradations when trying to avoid changes to the schema and settings. The design and an evaluation of the tool CloudAnalyst is presented in [9]. It is a visual modeller that utilizes the cloud simulation framework CloudSim [10] for analyzing cloud computing environments and applications. Different user bases, as well as various regions, data centers, applications, and workloads

can be modeled. A simulation can be used to estimate the operational costs. However, different application architecture candidates have to be modeled manually, no information concerning the structure of an existing software system can be applied automatically, and the cloud's utilized resources cannot be varied dynamically during a simulation run.

The authors in [11] contribute a performance and cost assessment of real cloud infrastructures. Here, the authors modeled a Service-Oriented Architecture (SOA) e-business application and two different workloads. Different pricing plans and hosting scenarios were modeled and implications on performance were evaluated as well. Moreover, platform limitations were explored on a coarse grained level. Our Cloud Environment Constraint (CEC) model (see Section IV) allows a more detailed view and automatic detection capabilities on a source code level compared to the system level used in that work. The authors determine a considerable potential for cutting costs running the modeled application in the cloud. However, they point out that "optimizing applications for very specific cost models may result in vendor lock in and a lack of flexibility and maintainability." In [12], the authors propose a conceptual cloud adoption toolkit that addresses the challenges of cloud adoption in enterprises. The toolkit provides five tools/techniques. Among those, the cost modeling tool utilizes Unified Modeling Language (UML) deployment diagrams to model an intended architecture for running existing software systems in a cloud environment. The deployment model is then augmented with price information that enables automated cost estimation for a specific cloud environment. In comparison, CloudMIG is intended to generate target architecture candidates for arbitrary cloud environments and to calculate the estimated costs for each deployment and in dependence of the observed or expected workload.

A profit-driven service request scheduling approach for clouds is described in [13]. Here, a particular focus is on a service provider and consumer perspective. Service requests have to be scheduled to satisfy the concerns of the service providers as well as the consumers. In this context, a pricing model and two profit-driven service request scheduling algorithms are presented. Linear programming is applied by the authors in [14] for addressing a task throughput maximization problem in a budget-constrained scenario.

Our CSA hierarchy evaluates the suitability and alignment of an existing software system with respect to a specific cloud environment. It focuses on technical opportunities and limitations and incorporates an automated detection of CEC violations (see Section IV). In contrast to that, the suitability index for the adoption of cloud computing technologies presented in [15] includes rather non-technical characteristics like the sensitivity of the system's data or its criticality. Nevertheless, it also considers the scale of existing IT resources and observed resource utilization patterns as well.

Table I. EUCALYPTUS HARDWARE CONFIGURATION

Component	Variant
CPU type	2x AMD Opteron 2384 2.7GHz (4 cores)
RAM	16 GB DDR2-667
Network	1 Gbit/s

Table II. VM INSTANCE TYPES

Name	#CPU cores	RAM (MB)
Standard.M	1	512
Standard.L	2	1,024
Memory.M	2	2,048
Memory.L	2	3,584
Compute.M	4	2,048
Compute.L	6	2,048

Table III. VM INSTANCE TYPE PRICE MODEL

VM instance type	Costs/hour (\$)
Standard.M	0.3
Standard.L	0.4
Memory.M	0.5
Memory.L	0.75
Compute.M	0.6
Compute.L	1.15

III. EXAMPLE SCENARIO

The experiment setup of our example scenario is described in Section III-A, the results are then presented in Section III-B.

A. Experiment Setup

We investigated the deployment of Apache OFBiz 9.04 into an installation of Eucalyptus. Apache OFBiz is a Java-based open source E-Commerce/ Enterprise Resource Planning (ERP) system. For instance, it provides several modules for accounting, order processing, and human resource management that are accessible via a web-based Graphical User Interface (GUI).

Eucalyptus is a cloud software for building private, hybrid, or public IaaS clouds. Its Application Programming Interface (API) is compatible with the popular Amazon EC2 and S3 services and it is also available in an open source version. Therefore, Eucalyptus is ideally suited for building cloud computing research test beds. The hardware listed in Table I was utilized for Eucalyptus' cluster and node controllers responsible for allocating and controlling the cluster of Virtual Machines (VMs). The superordinate cloud controller node was installed on an identically equipped machine. However, that second machine did not provide dedicated resources for VM allocation. In typical IaaS offerings as well as with Eucalyptus, a cloud user can choose between different VM instance types as basic building blocks. A VM instance type determines the hardware configuration that is available for running the user's virtual machine. With every start of a VM an appropriate instance type can be assigned according to the user's current needs.

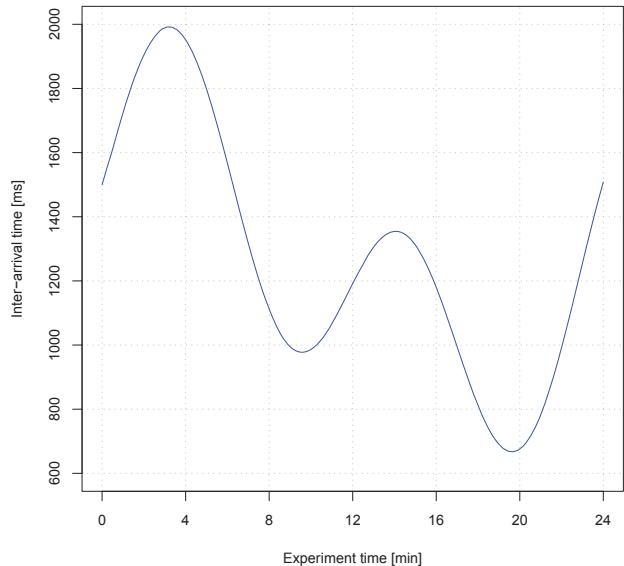


Figure 1. Inter-arrival time function.

To evaluate the implications of VM instance type selection we configured the six different VM instance types that are listed in Table II. In many cloud offerings, the VM instance types are priced on a pay-per-use basis and proportional to supplied resources (e.g., see [16], [17]). The selection of proper VM instance types may therefore have a substantial impact on overall operational costs. Since in real cloud offerings the amply equipped VM instance types are more costly by tendency, we used the price model shown in Table III that roughly follows this principle. The inter-arrival time function illustrated in Fig. 1 was applied to simulate a typical day night cycle usage pattern where the experiment minutes map to the hours of a day. Our employed user behavior emulated customers visiting the web store and browsing a product category. The number of user requests exhibits two peaks, one in the morning and one in the evening hours.

It should be noted that the demo installation of Apache OFBiz 9.04 was used that applies the rather slow embedded Java database Derby to deliver the demo catalog products. However, as the focus of our experiments was to compare the implications resulting from different VM instance types, this does not affect the results' validity. We were particularly interested in the resulting variations concerning the response times and the observed CPU utilizations. Regarding the response times we defined a limit of 1.5s that should not be exceeded for our test user sequence and which can be seen as a part of a virtual SLA [18]. As illustrated in Fig. 2, the usage of one single instance of a VM instance type was not always sufficient to fulfill the SLA. Here, one Standard.L instance provokes an SLA violation in the evening hours (Fig. 2 b). The single Standard.M instance (Fig. 2 a) exhibits an even more distinctive under-provisioning, as the CPU was

often used up to the full. As a consequence, Apache OFBiz repeatedly just returned error messages after experiencing a massive increase in response times up to minute 19, and therefore, caused the test to stop. Hence, in the following, we also investigated the minimum number of instances concerning each VM instance type that were necessary to satisfy the SLA. Here, we always maximized the Java Virtual Machine (JVM) heap size that could be configured according to the VM instance type specifications and that was available to Apache OFBiz.

B. Results

An overview regarding the measured response times and CPU utilizations following the varying load for each applied VM instance type is presented in Fig. 3. To stay below the 1.5s SLA response time limit, two instances of the Standard.M and Standard.L VM instance types were required in each case. The according Fig. 3a) and Fig. 3b) therefore show the average response times and CPU utilizations for both instances. The response times and CPU utilizations generally followed the usage pattern with a rise during peak times and exhibiting lower phases otherwise. Nevertheless, considering the response times this effect manifests more blurred for the aggregated measurements of the two Standard.M and Standard.L instances. Regarding the Standard.M instances the overall CPU utilization was still rather high. An interesting detail can be noticed in the Fig. 3c) - 3f) as there are short bursts around the 14th minute when the number of user requests leaves behind a local minimum.

Besides for the Standard.M VM instance type, the CPU utilizations fluctuate at a rather low level. Fig. 4 underlines this observation by showing the average CPU utilization for each experiment. Incorporating the Standard.M VM instance type, the avg. CPU utilizations range from 16%-59%, which translates to an avg. CPU over-provisioning ranging from 41%-84% at the same time. As mentioned before, we assume presence of a pay-per-use billing model. Considering our defined VM instance type price model (see Table III) the resulting operational costs being extrapolated for one month are presented in Fig. 5. Here, we simplifying presume that the usage pattern repeats each day and therefore the number of the minimally required instances remains stable. The cost minimum is reached by utilizing one Memory.M instance.

IV. CURRENT SHORTCOMINGS

The example scenario described in Section III reveals several general challenges considering the migration of software systems to a cloud environment. These shortcomings of the prevalent simplistic migration approaches form basic technical difficulties of cloud migration projects that need to be addressed by reengineers when migrating existing systems to the cloud and reworking them for optimized alignment. The example scenario emulates a common approach to minimize the migration effort and to obtain working results in a

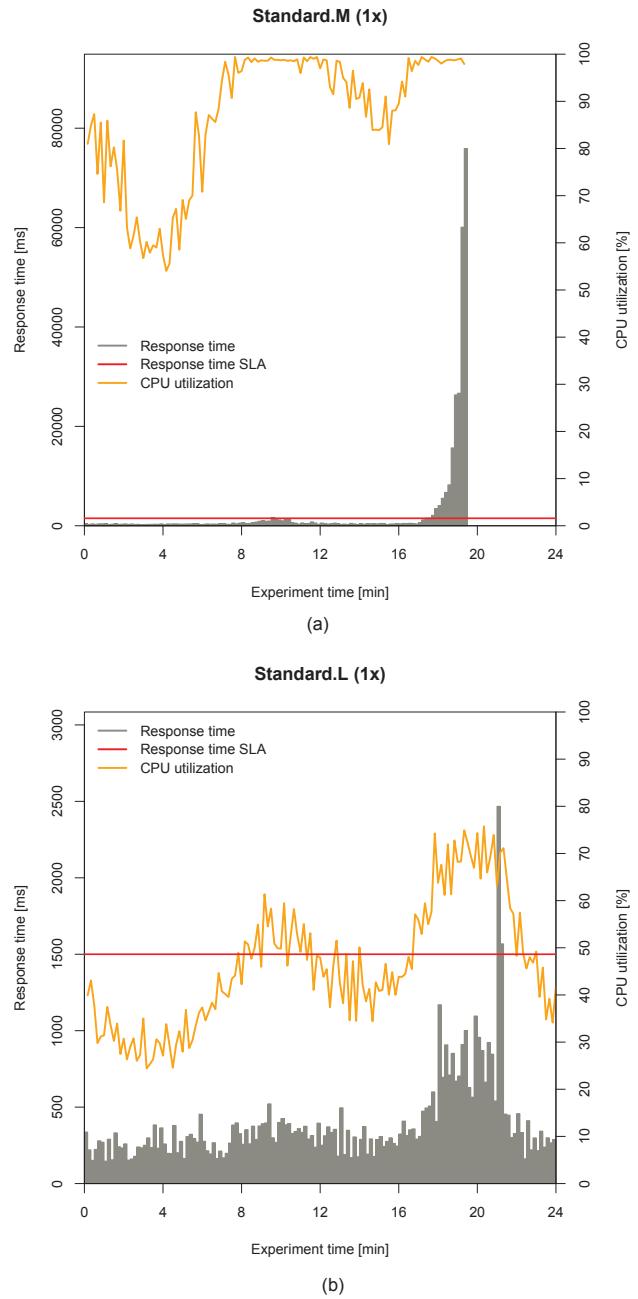


Figure 2. SLA violation when using a single instance of the Standard.M (a) or Standard.L (b) VM instance type.

short period. It deploys the regarding software system to coarse grained IaaS building blocks (VMs). After altering the persistency layer the existing system can be used in a cloud environment.

However, the experiments presented in Section III highlight many open issues. Running an existing application in the cloud does not imply relief of under- and over-provisioning concerns as such. Instead of supplying inappropriate physical on premise hardware configurations, the

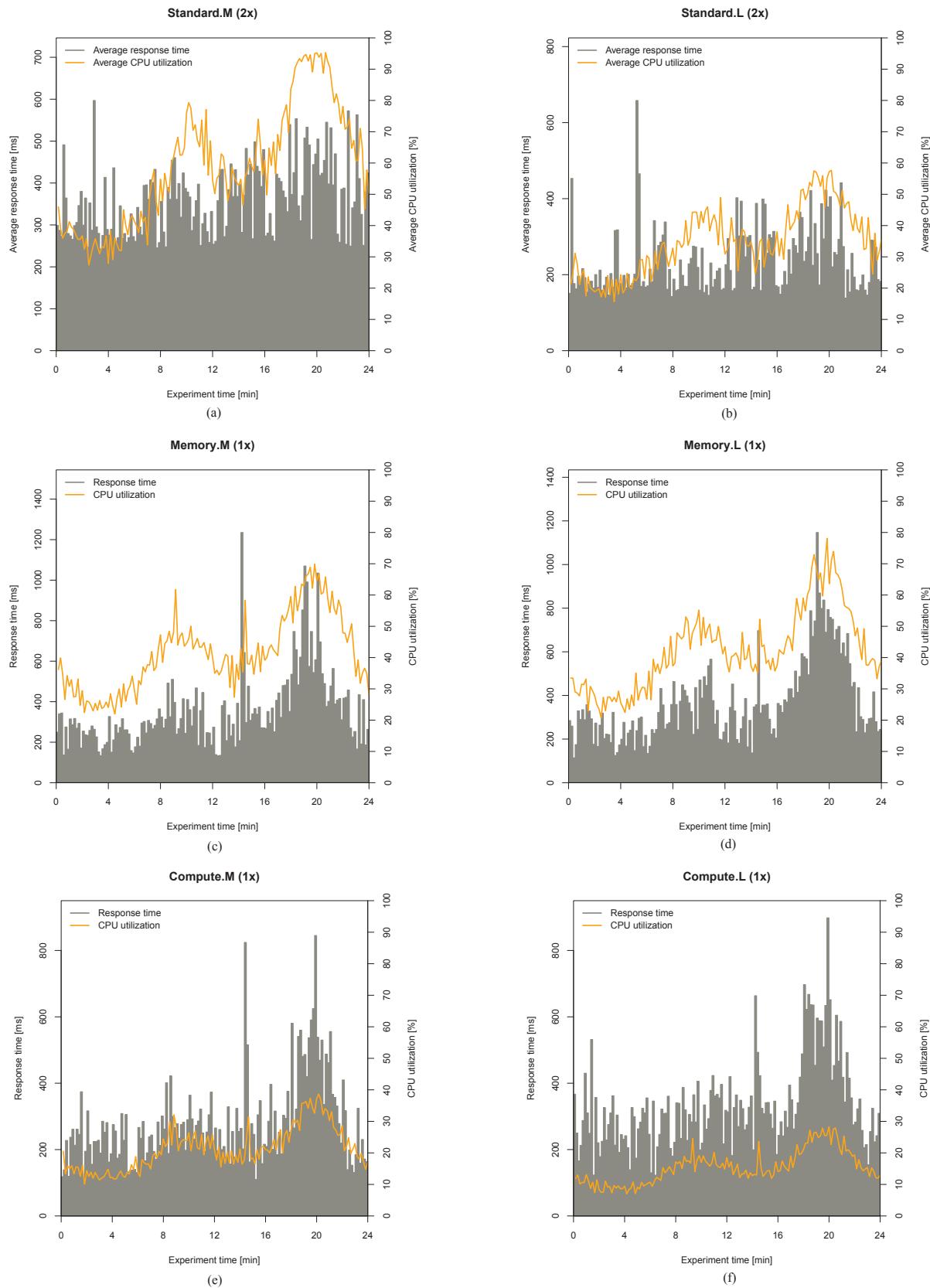


Figure 3. Response times and CPU utilizations for each VM instance type. Two instances were used for Standard.M (a) and Standard.L (b).

under- and over-provisioning of resources can easily be migrated to a cloud environment itself. For example, an inappropriate number of VM instances or unsuitable VM instance types could be employed. Fig. 2 demonstrates the resource under-provisioning in our example scenario. The hardware configuration of Standard.M and Standard.L VM instance types is too restricted for utilizing just a single instance. In this case the response times exceed the defined limit and cause a violation of the SLA. Moreover, this scenario shows the constrained scalability of an application running in a cloud environment. The operation in a cloud does not solve scalability issues per se. For example, an IaaS-based application often needs to have built-in self-adaptive capabilities for leveraging a cloud environment's elasticity. In contrast to the former example, Fig. 4 gives evidence for over-provisioning of cloud resources. Our experiments resulted in a maximum of average 84% over-provisioning of CPU resources for the Compute.L VM instance type implicating more than doubled operational costs compared to the possible minimum (see Fig. 5).

Nevertheless, the effects on additional expenditures cannot simply be evaluated according to the over-provisioning of resources. They depend on other factors as for example the selected VM instance type and do not necessarily scale linearly, as can be seen in Figs. 4 and 5, considering the Compute.M VM instance type in contrast. Comparing different cloud vendors would additionally complicate a cost estimation, as the different price models and VM instance type configurations impede assessment of real world usage scenarios as well. Hence, a better support for anticipating the operational costs without limiting the modeling capabilities to, for example, a set of specific cloud environments, a set of particular configurations, or resource types as VMs is needed. This is especially the case when incorporating PaaS cloud environments, which follow other design paradigms and offer basic building blocks that differ from the VMs used in IaaS-based clouds. Furthermore, our example scenario utilizes a repeating usage pattern as well as homogeneous VM instance types and a constant number of VM instances during an experiment run. This is likely to change in real world scenarios and adds additional complexity in evaluating migration alternatives and estimating the related costs. Further difficulties may arise considering architectural limitations of an existing system. For example, if distribution and parallelization is omitted in the present system design, there may emerge data inconsistency issues when scaling up horizontally while joining the VM instances to an existing data persistency layer. Moreover, exhibiting a reproducible short burst in response times after leaving behind a local minimum in the number of requests (see Section III-B), the experiments revealed an unexpected behavior of the application running in the cloud. In that regard, some effects may generally be hard to predict and therefore require profound evaluation.

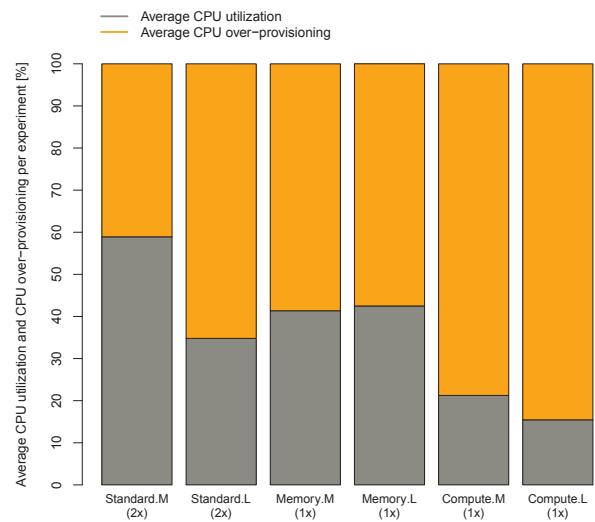


Figure 4. Average CPU utilization per conducted experiment. The statements in parentheses indicate the nr. of instances used for each VM instance type to satisfy the SLA.

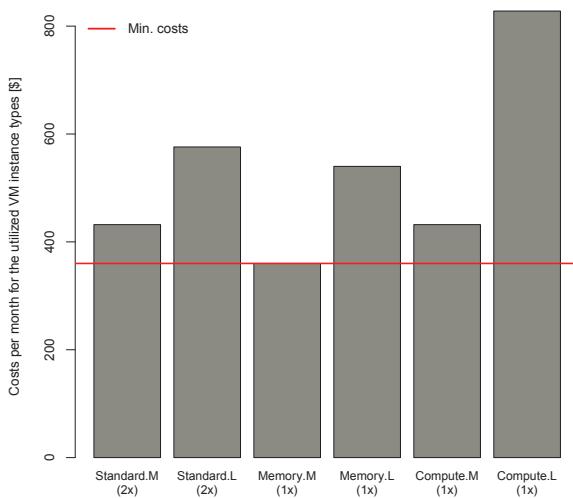


Figure 5. Extrapolated operational costs per month. The statements in parentheses indicate the nr. of instances used for each VM instance type to satisfy the SLA (included in calculation).

As mentioned before, the scalability issues as well as challenges regarding under- and over-provisioning are most often not solved by merely deploying an existing software system in a virtual machine and running it in an IaaS cloud environment. Therefore, we argue that migrating typical enterprise software to a cloud-based application usually implies an architectural restructuring step for aligning it with a cloud environment and exploit the cloud's offered advantages. However, knowledge about the internal structure

of an existing software system is often insufficient and therefore an architectural model has to be reconstructed first. The architectural model serves as a starting point for restructuring activities towards a cloud-optimized target architecture, which at the moment most often has to be created manually. This often is not an easy task, as construction of the advanced architecture usually presumes profound comprehension of the existing one. Furthermore, the target architecture must comply with the specific cloud environment's offered resources and imposed constraints, for example application frameworks and limitations of programming interfaces in PaaS cloud environments, respectively.

In this context, we introduced the notions of Cloud Environment Constraints (CECs), CEC violations, and CEC violation severities in [19] and [20]. For example, considering the cloud environment Google App Engine for Java a CEC would be the restriction of its sandbox environment that limits usage of Java Runtime Environment (JRE) types to only a subset of all types. A system that shall be migrated and that utilizes such an excluded type so far would raise a CEC violation. We defined the three CEC violation severities *Warning*, *Critical*, and *Breaking* that describe the likely effort for fixing a CEC violation, whereas the *Breaking* severity is most serious and causes the CloudMIG process to stop, for instance.

Besides the need for an automated detection of the CEC violations, a mapping model that describes the relationships between system parts of the status quo and a target architecture is required as well. Future workload in combination with the target architecture arrangement will determine resource utilization of the cloud environment during operation. As most cloud providers follow the paradigm of utility computing, and therefore, charge resource utilization on a pay-as-you-use basis, the arrangement of the target architecture has a direct impact on the operational costs.

To condense the difficulties and challenges described in this section, the shortcomings of today's simplistic migration approaches from typical enterprise software to cloud-based applications can be summarized as follows:

S1 Applicability: Solutions for migrating and aligning enterprise software to cloud-based applications are limited to particular cloud providers.

S2 Level of automation: To align existing systems with a cloud environment and to enable them to exploit the cloud's offered advantages, a reengineering step is required. Here, a target architecture and a mapping model currently often have to be built entirely manual. Additionally, the target architecture's violations against the cloud environment's constraints are not identified automatically at design time.

S3 Resource efficiency: Various migrated software systems are not designed to be resource-efficient and do

not leverage the cloud environments' elasticity, because even transferring an established application to a new cloud environment can be a cumbersome task itself. Over- and under-provisioning of resources is a challenge in cloud environments, too. Furthermore, means for evaluating a target architecture's dynamic resource utilization at design time are most often inadequate. This even strengthens the general problem that estimating the future operational costs for arbitrary cloud environments is difficult.

S4 Scalability: Scalability remains a concern in cloud environments as well. Automated support for evaluating a target architecture's scalability at design time is rare in the cloud computing context.

V. CSA HIERARCHY

To reason about the challenges emerging when migrating a specific system to a cloud environment and restructuring its architecture to facilitate a smooth integration into the cloud's service landscape, one has to judge the system's suitability upfront and the level of alignment with the cloud environment once the first steps are accomplished. To enable an evaluation and classification of software systems in this respect, we introduce the coarse grained Cloud Suitability and Alignment hierarchy (CSA hierarchy). As illustrated in Fig. 6, it comprises the five levels *cloud incompatible*, *cloud compatible*, *cloud ready*, *cloud aligned*, and *cloud optimized*. The levels are defined employing the notions of CECs, CEC violations, and associated CEC violation severities (see Section IV) and constitute revisited and modified applications of CloudMIG's workflow states explained in [19]. The five CSA hierarchy levels are being described in the following.

L0 Cloud incompatible: At least one CEC violation with severity *Breaking* exists.

L1 Cloud compatible: No CEC violations with severity *Breaking* exist.

L2 Cloud ready: No CEC violations exist.

L3 Cloud aligned: The execution context, utilized cloud services, or the migrated software system itself were configured to achieve an improved resource consumption (measurable in decreased costs that are to this effect charged by the cloud provider) or scalability without pervasively modifying the software system.

L4 Cloud optimized: The migrated software system was pervasively modified to enable automated exploitation of the cloud's elasticity. For example, its architecture was restructured to increase the level of parallelization. An evaluation was conducted to identify system parts which would experience an overall benefit from substitution or supplement with offered cloud services. These substitutions and supplements were performed.

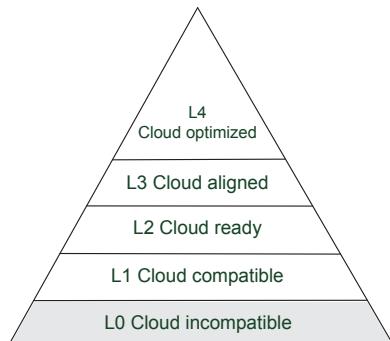


Figure 6. The CSA hierarchy.

The CSA hierarchy is “constructive” in its levels L1-L4. For example, for classifying a software system as *cloud aligned* it has to be *cloud compatible* and *cloud ready* as well. It should be noted that the CSA hierarchy solely considers technical concerns related to a migration to the cloud. In particular, it does not take organisational or economic restrictions into account, for example regarding governance issues, security policies, or a company’s business model. Regarding the example scenario in Section III, there exist no CEC violations that would impede proper execution after Apache OFBiz’s database is transferred to Eucalyptus’ persistent block storage, for instance. Concerning Eucalyptus, this activity is sufficient to lift Apache OFBiz 9.04 from *cloud compatible* to *cloud ready*. However, only through selecting the Memory.M VM instance type the application would be *cloud aligned* (see Fig. 5).

The CSA hierarchy defines the relationship of a specific configuration of a software system (e.g., regarding the version of the system’s software architecture) and a specific version of a cloud environment. A system being *cloud ready* concerning a specific cloud environment might be *cloud incompatible* regarding another one. Moreover, even for the same cloud environment this could change over time due to modifications of the incorporated cloud services offered by the cloud environment. Hence, the classification of a software system S regarding the CSA hierarchy depends on its configuration Θ and the cloud services Λ offered by a cloud environment. More specifically, the cloud environment provides n cloud services. A cloud service k is present in a particular version v : $\lambda_k^v \in \Lambda$. The classification of S regarding the CSA hierarchy level is then called Γ . Therefore, we can define a **CSA tuple** as follows:

$$\left(S, \Theta, \bigcup_k^n \lambda_k^v, \Gamma \right) \quad (1)$$

CSA tuples are utilized to compare cloud environment alternatives or competing software architectures when considering reengineering activities, for instance.

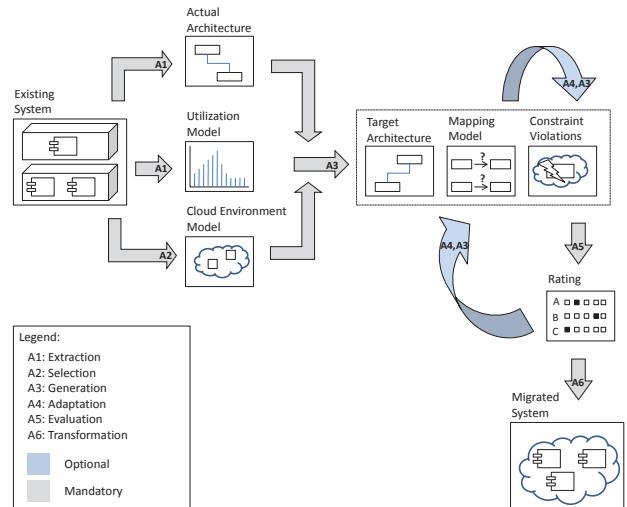


Figure 7. CloudMIG Overview.

VI. THE APPROACH CLOUDMIG

CloudMIG is composed of six activities for migrating an enterprise system to a cloud environment while addressing the shortcomings described in Section IV. It provides model-driven generation of considerable parts of the system’s target architecture. CEC violations are revealed automatically through analyzing an extracted system model. Furthermore, feedback loops allow for further alignment with the specific properties of the cloud environment and foster resource efficiency and scalability on an architectural level. Figure 7 outlines the approach. Its activities (A1-A6) are briefly described in the following, including the incorporated models.

A. Activity A1 - Extraction

CloudMIG aims at the migration of established enterprise applications. Usually, the architecture of software systems tends to erode over time. Therefore, initially envisioned architectures frequently diverge from actual implementations. The knowledge about the internal structure is often incomplete, erroneous, or even missing. As CloudMIG utilizes a model transformation during generation of its target architecture (cf. A3), a representation of the software system’s actual architecture has to be available first. Concerning this issue, an appropriate model is extracted by means of a software architecture reconstruction methodology. We propose OMG’s Knowledge Discovery Meta-Model (KDM) [21] for building a suitable meta-model.

For leveraging the commonly applied utility computing paradigm, the target architecture has to be laid out resource-efficient and elastic. Therefore, CloudMIG includes the extraction of an established software system’s utilization model acting as a starting point. The utilization model (resp. its meta-model) includes statistical properties concerning user behavior like service invocation rates over time or average submitted datagram sizes per request. Relevant

information can be retrieved from various sources. For example, considering log files or instrumenting the given system with our tool Kieker [22] for setting up a monitoring step constitute possible techniques. Furthermore, the utilization model contains application-inherent information related to proportional resource consumption. Metrics of interest could be a method's cyclomatic complexity or memory footprint. We propose OMG's Structured Metrics Meta-Model (SMM) [23] as a foundation for building the related meta-model.

B. Activity A2 - Selection

Common properties of different cloud environments are described in a Cloud Environment Model (CEM) [19]. Selecting a cloud provider specific environment as a target platform for the migration activities therefore implies the selection of a specific instance of the CEM. For example, the CEM comprises entities like VM instances or worker threads for IaaS and PaaS-based cloud environments, respectively. As a result, for every cloud environment, which shall be targeted with CloudMIG, a corresponding instance of CEM has to be created once beforehand. Transformation rules define possible relationships to the architecture meta-model.

We plan to attach further information related to scalability issues to the included entities, which can be configured by the reengineer in activity A4. For example, VM instances could provide hooks for controlling their lifetime dependent on dynamic resource utilization during runtime. Furthermore, the CEM includes constraints imposed by cloud environments restricting the reengineering activities (CECs). For example, the opening of sockets or the access to the file system are often constrained.

C. Activity A3 - Generation

The generation activity produces three artefacts, namely a target architecture, a mapping model, and a model characterizing the target architecture's violations of the cloud environment constraints. The latter lists the CEC violations and results in the construction of an initial CSA tuple. These constraint violations explicitly highlight the target architecture's parts which have to be redesigned manually by the reengineer (cf. A6). The mapping model assigns elements from the actual architecture to those included in the target architecture. Finally, the target architecture constitutes a primary artefact. It is realized as an instance of the CEM, which embeds this model. We propose the three phases P1-P3 for the generation of the target architecture that are illustrated in Figure 8. The phases are constructed as follows.

P1 - Model transformation: The phase P1 produces an initial assignment from elements of the existing architecture to cloud-specific elements available in the CEM. The initial

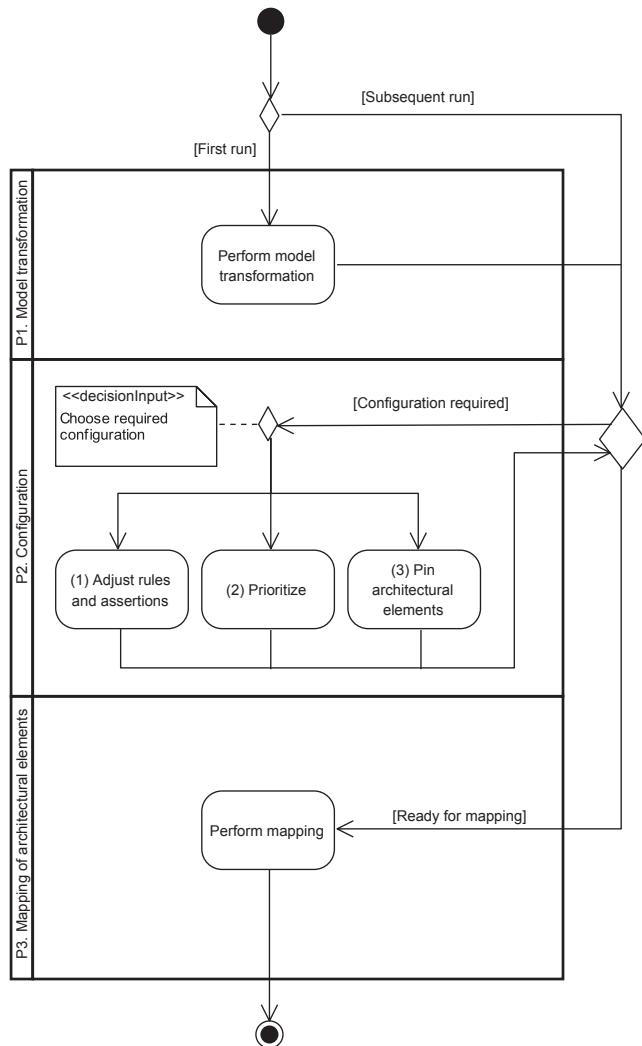


Figure 8. Target architecture generation process.

assignment is created applying a model-to-model transformation according to the transformation rules included in the cloud environment model (cf. activity A2).

P2 - Configuration: The phase P2 serves as a configuration of the algorithm used for obtaining a mapping of architectural elements in the phase P3. During P2, a reengineer may adjust rules and assertions for heuristic computation (cf. P3). A rule could be formulated like the following examples: “Distribute the five most frequently used services to own virtual machines” or “The server methods responsible for at least 10% of overall consumption of the CPU time shall be moved to client side components if they do not need access to the database”. An exemplary assertion could be: “An existing component must not be divided in more than 3 resulting components”. It is intended to provide a set of default rules and assertions. In addition to that, the reengineer will be given the possibility to modify them either via altering

the regarding numerical values or applying a corresponding domain-specific language (DSL). In both cases, the rules and assertions have to be prioritized after their selection. Hereby, the reengineer determines their significance during execution of P3. This means that architectural elements which are related to higher-weighted rules will be considered priorly for assignment and therefore have a stronger impact on the further composition of the target architecture. Furthermore, a reengineer may pin architectural elements. This prevents the rearrangement of previously assigned architectural elements to other target architecture components in phase P3.

P3 - Mapping of architectural elements: The phase P3 improves the initial assignment of architectural elements generated in phase P1 referring to resource-efficiency. Therefore, the formulated rules are utilized and the compliance of the resulting architecture with the defined assertions is considered. There exists an enormous number of possible combinations for assigning architectural elements. Efficiency improvements for one resource can lead to degradation for other resources or impair some design quality attributes. For example, splitting a component's parts towards different virtual machines can improve relative CPU utilization, but may lead to increased network traffic for intra-component communication and a decreased cohesion. Additionally, those effects do not necessarily have to move on linearly and moreover, the interrelations are often ambiguous as well. Therefore, we propose application of a heuristic rule-based approach to achieve an overall improvement. A potential algorithm is sketched in Listing 1 and it works as follows.

The rules are considered successively according to their priority. Thus, rules with higher priorities are weighted higher and have a stronger impact on the generated target architecture. The selection criterion of a rule is defined to deliver a set of scalar architectural elements. All possible subsets of the set are rated respective to the quality of the target architecture that would result, if the elements in the subset would be assigned correspondingly. This aims at considering interdependencies at the level of a single rule. For regarding interdependencies on an inter-rule level, the formulated assertions are taken into account. A rule is only applied if the reengineer did not formulate an assertion with a higher priority that would be violated after the rule's execution. Furthermore, the rule is applied to all mentioned subsets in order of their score. However, the rule is only utilized if no rearrangement of elements is necessary whose subset was rated higher. The same applies to assignments that would lead to rearrangement of elements that were placed by rules of higher priority or formerly pinned elements.

D. Activity A4 - Adaptation

The activity A4 allows the reengineer to manually adjust the target architecture towards case-specific requirements

```

1:  $E_{Pinned} \leftarrow$  Pinned architectural elements
2:  $R \leftarrow$  All rules
3:  $A \leftarrow$  All assertions
4:  $R_{Sort} \leftarrow$  Sort  $R$  descending by priority
5:  $E_{AllAffected} \leftarrow E_{Pinned}$ 
6: for all  $r$  in  $R_{Sort}$  do
7:    $E_r \leftarrow$  All architectural elements delivered by  $r$ 's
      selection criterion
8:    $P_r^E \leftarrow$  Power set of  $E_r$ 
9:    $Score \leftarrow$  New associative array
10:  for all  $p_r^E$  in  $P_r^E$  do
11:     $Score[p_r^E] \leftarrow$  Rate  $p_r^E$ 
12:  end for
13:   $Score_{Sort} \leftarrow$  Sort  $Score$  descending by score
14:   $Score_{Sort}^{Keys} \leftarrow$  Keys of  $Score_{Sort}$ 
15:  for all  $p_r^E$  in  $Score_{Sort}^{Keys}$  do
16:     $E_{FormerlyAffected} \leftarrow p_r^E \cap E_{AllAffected}$ 
17:     $E_{NeedReassignment} \leftarrow$  Elements of
       $E_{FormerlyAffected}$  that need reassignment
      conc.  $r$ 
18:    if  $E_{NeedReassignment} == \emptyset$  then
19:       $A_{HigherPrio} \leftarrow$  All  $a \in A$  with higher priority
      than  $r$ 
20:      if  $\nexists a \in A_{HigherPrio}$  with  $r$  violates  $a$  then
21:        Apply rule  $r$  to all elements in  $p_r^E$ 
22:         $E_{AllAffected} = E_{AllAffected} \cup p_r^E$ 
23:      end if
24:    end if
25:  end for
26: end for

```

Listing 1. Rule-based heuristics for creating a mapping of architectural elements that improves resource efficiency.

that could not be fulfilled during generation activity A3. For example, the generation process might not have yielded an expected assignment of a critical component. Furthermore, for leveraging the elasticity of a cloud environment, the reengineer might configure a capacity management strategy by means of utilizing the hooks provided by entities contained in the CEM (cf. A2).

E. Activity A5 - Evaluation

For being able to judge about the produced target architecture and the configured capacity management strategy, A5 evaluates the outcomes of the activities A3 and A4. The evaluation involves static and dynamic analyses of the target architecture. The results can be aggregated in a CSA tuple. For example, metrics as LCOM or WMC can be utilized for static analyses. Considering the target architecture's expected runtime behavior, we propose to apply a simulation on the basis of CloudSim. Thus, we intend to contribute a transformation from CloudMIG's CEM to CloudSim's simulation model.

F. Activity A6 - Transformation

This activity comprises the actual transformation of the enterprise system from the generated and improved target architecture to the aimed cloud environment. No further support for actually accomplishing the implementation is planned at this time.

VII. CONCLUSION AND FUTURE WORK

We presented an overview concerning our model-based approach CloudMIG for migrating legacy software systems to scalable and resource-efficient cloud-based applications. It concentrates on the SaaS provider perspective and facilitates the migration of enterprise software systems towards generic IaaS and PaaS-based cloud environments. We argued for explicit reengineering activities during the migration and motivated them based on experiments we conducted using the cloud software Eucalyptus and the e-commerce/ ERP system Apache OFBiz. Our example scenario demonstrated some of the limitations regarding the currently prevalent simplistic migration approaches. Considering the reengineering activities, CloudMIG is intended to generate considerable parts of a resource-efficient target architecture utilizing a rule-based heuristics. To classify the suitability of cloud environments for given systems and the degree of alignment during a reengineering process, we introduced the CSA hierarchy. The future work focuses on the realization, improvement, and evaluation of CloudMIG's target architecture generation and evaluation activities (A3 and A5).

REFERENCES

- [1] S. Frey and W. Hasselbring, "Model-Based Migration of Legacy Software Systems to Scalable and Resource-Efficient Cloud-Based Applications: The CloudMIG Approach," in *Proceedings of the First International Conference on Cloud Computing, GRIDs, and Virtualization (Cloud Computing 2010)*, Lisbon, Portugal, Nov. 2010, pp. 155–158.
- [2] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "The Eucalyptus Open-Source Cloud-Computing System," in *Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, 2009. CCGRID '09*, May 2009, pp. 124–131.
- [3] The Apache Software Foundation, "The Apache Open For Business Project (Apache OFBiz)," <http://ofbiz.apache.org/>, (Accessed January 20, 2012).
- [4] L. Wu, H. Sahraoui, and P. Valtchev, "Coping with legacy system migration complexity," in *Proceedings. 10th IEEE International Conference on Engineering of Complex Computer Systems, 2005. ICECCS 2005*, Jun. 2005, pp. 600–609.
- [5] I. Sriram and A. Khajeh-Hosseini, "Research Agenda in Cloud Technologies," *CoRR*, vol. abs/1001.3259, 2010.
- [6] T. Dillon, C. Wu, and E. Chang, "Cloud Computing: Issues and Challenges," in *Proceedings of the 24th IEEE International Conference on Advanced Information Networking and Applications (AINA), 2010*, pp. 27–33.
- [7] A. Khajeh-Hosseini, D. Greenwood, and I. Sommerville, "Cloud Migration: A Case Study of Migrating an Enterprise IT System to IaaS," *CoRR*, vol. abs/1002.3492, 2010.
- [8] A. Thakar and A. Szalay, "Migrating a (Large) Science Database to the Cloud," in *HPDC '10: Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*. New York, NY, USA: ACM, 2010, pp. 430–434.
- [9] B. Wickremasinghe, R. Calheiros, and R. Buyya, "CloudAnalyst: A CloudSim-Based Visual Modeller for Analysing Cloud Computing Environments and Applications," in *Proceedings of the 24th IEEE International Conference on Advanced Information Networking and Applications (AINA), 2010*, pp. 446–452.
- [10] R. N. Calheiros, R. Ranjan, C. A. F. D. Rose, and R. Buyya, "CloudSim: A Novel Framework for Modeling and Simulation of Cloud Computing Infrastructures and Services," *CoRR*, vol. abs/0903.2525, 2009.
- [11] P. Brebner and A. Liu, "Performance and Cost Assessment of Cloud Services," in *Service-Oriented Computing*, ser. Lecture Notes in Computer Science, E. Maximilien, G. Rossi, S.-T. Yuan, H. Ludwig, and M. Fantinato, Eds. Springer Berlin/ Heidelberg, 2011, vol. 6568, pp. 39–50.
- [12] D. Greenwood, A. Khajeh-Hosseini, J. W. Smith, and I. Sommerville, "The Cloud Adoption Toolkit: Addressing the Challenges of Cloud Adoption in Enterprise," *CoRR*, vol. abs/1003.3866, 2010.
- [13] Y. C. Lee, C. Wang, A. Y. Zomaya, and B. B. Zhou, "Profit-Driven Service Request Scheduling in Clouds," in *Proceedings of the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid)*, May 2010, pp. 15–24.
- [14] W. Shi and B. Hong, "Resource Allocation with a Budget Constraint for Computing Independent Tasks in the Cloud," in *Proceedings of the IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom)*, Dec. 2010, pp. 327–334.
- [15] S. C. Misra and A. Mondal, "Identification of a company's suitability for the adoption of cloud computing and modelling its corresponding Return on Investment," *Mathematical and Computer Modelling*, vol. 53, no. 3-4, pp. 504–521, 2011.
- [16] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the Clouds: A Berkeley View of Cloud Computing," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28, Feb. 2009.
- [17] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, "A break in the clouds: towards a cloud definition," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 1, pp. 50–55, 2009.
- [18] W. Iqbal, M. Dailey, and D. Carrera, "SLA-Driven Adaptive Resource Management for Web Applications on a Heterogeneous Compute Cloud," in *CloudCom*, ser. Lecture Notes in Computer Science, M. G. Jaatun, G. Zhao, and C. Rong, Eds., vol. 5931. Springer, 2009, pp. 243–253.

- [19] S. Frey and W. Hasselbring, "An Extensible Architecture for Detecting Violations of a Cloud Environment's Constraints During Legacy Software System Migration," in *Proceedings of the 15th European Conference on Software Maintenance and Reengineering (CSMR 2011)*, T. Mens, Y. Kanellopoulos, and A. Winter, Eds. IEEE Computer Society, Mar. 2011, pp. 269–278.
- [20] S. Frey, W. Hasselbring, and B. Schnoor, "Automatic conformance checking for migrating software systems to cloud infrastructures and platforms," *Journal of Software Maintenance and Evolution: Research and Practice*, doi: 10.1002/smrv.582, 2012.
- [21] Object Management Group, Inc., "Architecture-Driven Modernization (ADM): Knowledge Discovery Metamodel (KDM), V. 1.3," <http://www.omg.org/spec/KDM/>, (Accessed January 20, 2012).
- [22] A. van Hoorn, M. Rohr, W. Hasselbring, J. Waller, J. Ehlers, S. Frey, and D. Kieselhorst, "Continuous monitoring of software services: Design and application of the Kieker framework," Department of Computer Science, University of Kiel, Germany, Tech. Rep. TR-0921, Nov. 2009.
- [23] Object Management Group, Inc., "Architecture-Driven Modernization (ADM): Structured Metrics Meta-Model (SMM), V. 1.0 Beta 3," <http://www.omg.org/spec/SMM/>, (Accessed January 20, 2012).

A Formal Language for the Expression of Pattern Compositions

Ian Bayley and Hong Zhu

Department of Computing and Communication Technologies

Oxford Brookes University

Oxford OX33 1HX, UK.

Email: ibayley@brookes.ac.uk, hzh@brookes.ac.uk

Abstract—In real applications, design patterns are almost always to be found composed with each other. Correct application of patterns therefore relies on precise definition of these compositions. In this paper, we propose a set of operators on patterns that can be used in such definitions. These operators are restriction of a pattern with respect to a constraint, superposition of two patterns, and a number of structural manipulations of the pattern's components. We demonstrate the uses of these operators by examples. We also report a case study on the pattern compositions suggested informally in the Gang of Four book in order to demonstrate the expressiveness of the operators.

Keywords-Design patterns, Pattern composition, Object oriented design, Formal methods.

I. INTRODUCTION

As codified reusable solutions to recurring design problems, design patterns play an increasingly important role in the development of software systems [2], [3]. In the past few years, many such patterns have been identified, catalogued [2]–[15], formally specified [16]–[20], and included in software tools [21]–[31]. Although each pattern is specified separately, they are usually to be found composed with each other in real applications. It is therefore vital to represent pattern compositions precisely and formally, so that the correct usage of composed patterns can be verified and validated.

The composition of design patterns have been studied by many authors informally, e.g., in [32], [33]. Visual notations such as the *Pattern:Role* annotation, and a forebear based on Venn diagrams, have been proposed by Vlissides [34] and widely used in practice. They indicate where, in a design, patterns have been applied so their compositions are comprehensible. These notations focus on static properties. In [35], Dong *et al.* developed techniques for visualising pattern compositions in such notations by defining appropriate UML profiles. Their tool, deployed as a web service, identifies pattern applications, and does so by displaying stereotypes, tagged values, and constraints. Such information is delivered dynamically with the movement of the user's mouse cursor

This paper is an extended and revised version of the paper [1] presented at the 2nd International Conference on Pervasive Patterns and Applications (PATTERNS 2011)

on the screen. Their experiments show that this delivery on demand helps to reduce the information overload faced by designers.

More recently, Smith proposed the *Pattern Instance Notation* (PIN), to visually represent the composition of patterns in a hierarchical manner [36]. Most importantly, he also recognised that multiple instances of roles needed to be better expressed and he devised a suitable graphic notation for this. However, while many approaches to pattern formalisation have been proposed, very few authors have investigated pattern composition formally. Two of those who have are Dong *et al.* [37]–[41] and Taibi and Ngo [18], [42], [43], respectively.

As far as we know, Dong *et al.* were the first to study pattern composition in a formal setting [37]. In their approach, a composition of two patterns is defined as a pair of *name mappings*. Each mapping "associates the names of the classes and objects declared in a pattern with the classes and objects declared in the composition of this pattern and other patterns" [37]. They illustrate this by composing Composite with Iterator [37]–[39]. Dong *et al.* also demonstrated that how structural and behavioural properties of the instances of patterns and their compositions can be inferred from their formal specifications.

In [41], they developed this approach further recently in their study on the commutability of pattern instantiation with pattern integration, another term for pattern composition. A pattern instantiation was defined as a mapping from names of various kinds of elements in the pattern to classes, attributes, methods, *etc.*, in the instance. An integration of two patterns was defined as a mapping from the set union of the names of the elements in the two patterns into the names of the elements in the resulting pattern. However, in a recent study of the compositions of security patterns [40], they merely presented the compositions in the form of diagrams, from which they manually derived the formal specifications afterwards.

Taibi and Ngo [43] took an approach very similar to this, but instead of defining mappings for pattern compositions and instantiations, they use substitution to directly rename the variables that represent pattern elements. Instantiation replaces these variables with constants, whereas composition

replaces them with new variables, before then combining the predicates. They illustrated the approach by combining the Mediator and Observer patterns in [42] and the Command and Composite patterns in [43].

In [44], we formally defined a pattern composition operator based on the notion of overlaps between the elements in the composed patterns. We distinguished three different kinds of overlaps: one-to-one, one-to-many and many-to-many. The compositions in Dong *et al.* and Taibi's approaches all have overlaps that are one-to-one. However, the other two kinds are often required. For example, if the Composite pattern is composed with the Adapter pattern in such a way that one or more of the leaves are adapted then that is a one-to-many overlap. This cannot be represented as a mapping between names, nor by a substitution or instantiation of variables. However, although our overlap based operator is universally applicable, we found in our case study that it is not very flexible for practical uses and its properties are complex to analyse.

In this paper, therefore, we revise our previous work and take a radically different approach. Instead of defining a single universal composition operator, we propose a set of six more primitive operators, with which each sort of composition can then be accurately and precisely expressed. This paper makes the following three the main contributions.

- A set of operators on design patterns are formally defined.
- The uses of the operators in pattern-based software design are illustrated by classic examples in the literature.
- The expressiveness of the operators is demonstrated by a case study on the compositions of the patterns suggested by the Gang of Four book [2].

The remainder of the paper is organised as follows. Section II provides a background by reviewing the different approaches to pattern formalisation. Section III formally defines the six operators. Section IV gives two examples to illustrate how compositions can now be specified. Section V reports a case study in which we used the operators to realise all the pattern combinations suggested by the Gang of Four (GoF) book [2]. Section VI concludes the paper with a discussion of related works and future work.

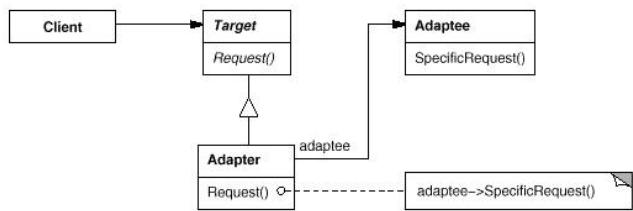
II. BACKGROUND

In the past few years, researchers have advanced several approaches to the formalisation of design patterns. In spite of differences in these formalisms, the basic underlying ideas are quite similar. In particular, valid pattern instances are usually specified using statements that constrain their structural features and sometimes their behavioural features too. The structural constraints are typically assertions that certain types of components exist and have a certain static configuration. The behavioural constraints, on the other hand, detail the temporal order of messages exchanged between the components that realise the designs.

The various approaches to pattern formalisation differ in how they represent software systems and in how they formalise the predicate. For example, Eden's predicates are on the source code of object-oriented programs [19] but they are limited to structural features. Taibi's approach in [18] is similar but he takes the further step of adding temporal logic for behavioural features. In contrast, our predicates are built up from primitive predicates on UML class and sequence diagrams [20]. These primitives are induced from GEBNF (*Graphic Extension of Backus-Naur Form*) definition of the abstract syntax of graphical modelling languages [45], [46]. Nevertheless, the operators on design patterns used in this paper are generally applicable and independent of the particular formalism used. Still, the example specifications of GoF patterns come from our previous work [20].

As examples, Figures 1 and 2 show the specifications of the Object Adapter and Composite design patterns, respectively. The class diagrams from the GoF book have been reproduced to enhance readability; while their sequence diagrams are omitted for the sake of space. The primitive predicates and functions we use are explained in Table I. All of them are either induced directly from the GEBNF definition of UML, or are defined formally in terms of such predicates. The predicate *trigs* is particularly important in describing dynamic behavioural properties and it is formally defined as follows.

$$\begin{aligned} \text{trigs}(m, m') \triangleq \\ \text{toAct}(m) = \text{fromAct}(m') \wedge m < m' \end{aligned}$$



Specification 1: (Object Adapter Pattern) Components

- 1) $\text{Client}, \text{Target}, \text{Adapter}, \text{Adaptee} \in \text{classes}$,
- 2) $\text{requests}, \text{specreqs} \subseteq \text{operations}$,

Dynamic Components

- 1) $mr, ms \in \text{messages}$

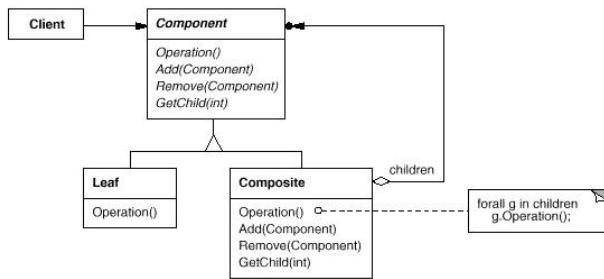
Static Conditions

- 1) $\text{requests} \subseteq \text{Target.ops}$,
- 2) $\text{specreqs} \subseteq \text{Adaptee.ops}$
- 3) $\text{Adapter} \rightarrow \text{Target}$,
- 4) $\text{Adapter} \rightarrow \text{Adaptee}$,
- 5) $\text{Client} \rightarrow \text{Target}$

Dynamic Conditions

- 1) $mr.sig \in \text{requests}$
- 2) $ms.sig \in \text{specreqs}$
- 3) $\text{trigs}(mr, ms)$

Figure 1. Specification of Object Adapter Pattern



Specification 2: (Composite) Components

- 1) Client, Component, Leaf, Composite \in classes
- 2) operation \in operations

Dynamic Components

- 1) $m_1, m_2 \in$ messages

Static Conditions

- 1) operation \in Component.ops
- 2) Leaf \rightarrow Component
- 3) Composite \rightarrow Component
- 4) Client \rightarrow Component
- 5) Composite $\diamond \rightarrow^*$ Component
- 6) \neg Leaf $\diamond \rightarrow^*$ Component
- 7) operation.isAbstract

Dynamic Conditions

- 1) $m_1.sig =$ Composite.operation
- 2) isOp(m_2)
- 3) trigs(m_1, m_2)
- 4) $m_2.sig =$ Leaf.operation \Rightarrow
 $\neg \exists m_3 \in$ messages \cdot trigs(m_2, m_3) \wedge isOp(m_3)

Figure 2. Specification of Composite Pattern

The definition of the Composite pattern uses an auxiliary predicate *isOp* defined on messages as follows.

$$\begin{aligned} \text{isOp}(m) \triangleq \\ m.sig = \text{Leaf.operation} \vee \\ m.sig = \text{Composite.operation} \end{aligned}$$

In general, a design pattern P can be defined abstractly as an ordered pair $\langle V, Pr \rangle$, where Pr is a predicate on the domain of some representation of software systems, and V is a set of declarations of variables free in Pr . In other words, Pr specifies the structural and behavioural features of the pattern and V specifies its components. Let $V = \{v_1 : T_1, \dots, v_n : T_n\}$, where v_i are variables that range over the type T_i of software elements. The semantics of the specification is a ground predicate in the following form.

$$\exists v_1 : T_1 \dots \exists v_n : T_n \cdot (Pr) \quad (1)$$

Note that, for the sake of readability, in the examples we split the predicate in the specification into two parts: one for static conditions and the other for dynamic conditions as in [16], [18], [37] and [20]. In the sequel, we write $Spec(P)$

Table I
THE FUNCTIONS AND PREDICATES USED IN THE EXAMPLES

ID	Meaning
<i>classes</i>	The set of class nodes in the class diagram
<i>operations</i>	The set of operations in the class diagram
<i>C.ops</i>	The operations contained in the class node C
<i>m.sig</i>	The signature of the message m as operation
$X \rightarrow Y$	Class X inherits class Y directly or indirectly
$X \rightarrow Y$	There is an association (either direct or indirect) from class X to Y
$X \diamond \rightarrow Y$	There is an composite or aggregate relation (either direct or indirect) from X to Y
<i>C.op</i>	The redefinition of op for class C
<i>trigs(m, m')</i>	Message m is sent to the activation from which message m' is afterwards sent
<i>isAbstract(C)</i>	Class C is abstract
<i>op.isAbstract</i>	Operation op is abstract
<i>fromLL(m)</i>	The lifeline from which message m is sent
<i>toLL(m)</i>	The lifeline to which message m is sent
<i>l.class</i>	The class of the lifelines
<i>hasParam(m, p)</i>	p is one of the parameters of message m
<i>returnValue(m)</i>	The value returned by message m

to denote the predicate (1) above, $Vars(P)$ for the set of variables declared in V , and $Pred(P)$ for the predicate Pr .

Note further that the above definition can easily be generalised or adapted so that the predicates in pattern specifications are defined on the domain of program implementations and their dynamic behaviours.

We can formally define the conformance of a design model m to a pattern P , written as $m \models P$, and reason about the properties of instances based on the patterns they conform to, but we omit the details here for the sake of space. Readers are referred to [20] and [45]. The theory developed in this paper remains valid so long as this notion of conformance is valid and the logic is consistent. However, for the sake of simplicity, this paper only considers designs represented as models.

III. OPERATORS ON PATTERNS

We now formally define the operators on design patterns.

A. Restriction operator

The restriction operator was first introduced in our previous work [44], where it is called the *specialisation* operator.

Definition 1: (Restriction operator)

Let P be a given pattern and c be a predicate defined on the components of P . A restriction of P with constraint c , written as $P[c]$, is the pattern obtained from P by imposing the predicate c as an additional condition on the pattern. Formally,

- 1) $Vars(P[c]) = Vars(P)$,
- 2) $Pred(P[c]) = (Pred(P) \wedge c)$. \square

For example, a variant of the Adapter pattern in which there is only one request and one specific request, hereafter known as *Adapter*₁, can be formally defined as follows.

$Adapter_1 \triangleq$

$$Adapter[||requests|| = 1 \wedge ||specreqs|| = 1].$$

Restriction is frequently used in the case study, particularly in the form $P[u = v]$ for pattern P and variables u and v of the same type. This expression denotes the pattern obtained from P by unifying u and v to make them the same element.

Note that the instantiation of a variable u in pattern P with a constant a of the same type of variable u can also be expressed by using restriction: $P[u = a]$.

This operator does not introduce any new components into the structure of a pattern, but the following operators do.

B. Superposition operator

Definition 2: (Superposition operator)

Let P and Q be two patterns. Assume that the component variables of P and Q are disjoint, i.e., $Vars(P) \cap Vars(Q) = \emptyset$. The *superposition* of P and Q , written $P * Q$, is a pattern that consists of both pattern P and pattern Q as formally defined below.

- 1) $Vars(P * Q) = Vars(P) \cup Vars(Q);$
- 2) $Pred(P * Q) = Pred(P) \wedge Pred(Q).$ \square

For example, the superposition of Composite and Adapter patterns, *Composite * Adapter*, requires each instance to contain one part that satisfies the Composite pattern and another that satisfies the Adapter pattern. These parts may or may not overlap, but the following expression does enforce an overlap, as it requires that the *Leaf* class be the target of an Adapter.

$(Composite * Adapter)[Target = Leaf]$

The requirement that $Vars(P)$ and $Vars(Q)$ be disjoint is easy to fulfil using renaming. An appropriate notation for this will be introduced later.

C. Extension operator

Definition 3: (Extension operator)

Let P be a pattern, V be a set of variable declarations that are disjoint with P 's component variables (i.e., $Vars(P) \cap V = \emptyset$), and c be a predicate with variables in $Vars(P) \cup V$. The extension of pattern P with components V and linkage condition c , written as $P\#(V \bullet c)$, is defined as follows.

- 1) $Vars(P\#(V \bullet c)) = Vars(P) \cup V;$
- 2) $Pred(P\#(V \bullet c)) = Pred(P) \wedge c.$ \square

D. Flatten operator

Definition 4: (Flatten Operator)

Let P be a pattern, $(xs : \mathbb{P}(T)) \in Vars(P)$, $x \notin Vars(P)$, and $Pred(P) = p(xs, x_1, \dots, x_k)$. The flattening of P on variable xs , written $P \Downarrow xs \setminus x$, is the pattern defined as follows:

- 1) $Vars(P \Downarrow xs \setminus x) =$

$$(Vars(P) - \{(xs : \mathbb{P}(T))\}) \cup \{x : T\};$$

- 2) $Pred(P \Downarrow xs \setminus x) = p(\{x\}, x_1, \dots, x_k).$

Note that $\mathbb{P}(T)$ denotes the power set of T . For example, in the specification of the Adapter pattern, the component variable *requests* is a subset of *operations* so its type is $\mathbb{P}(\text{operation})$.

The single-leaf variant of the Adapter pattern $Adapter_1$ can also be defined as follows.

$Adapter_1 \triangleq$

$$(Adapter \Downarrow requests \setminus request) \Downarrow specreq \setminus specreqs$$

As an immediate consequence of this definition, we have the following property. For $x_1 \neq x_2$ and $xs_1 \neq xs_2$,

$$(P \Downarrow xs_1 \setminus x_1) \Downarrow xs_2 \setminus x_2 = (P \Downarrow xs_2 \setminus x_2) \Downarrow xs_1 \setminus x_1. \quad (2)$$

Therefore, we can overload the \Downarrow operator to a set of component variables. Formally, let XS be a subset of P 's component variables all of power set type, i.e., $XS = \{xs_1 : \mathbb{P}(T_1), \dots, xs_n : \mathbb{P}(T_n)\} \subseteq Vars(P)$, $n \geq 1$ and $X = \{x_1 : T_1, \dots, x_n : T_n\} \cap Vars(P) = \emptyset$, we write $P \Downarrow XS \setminus X$ to denote $P \Downarrow xs_1 \setminus x_1 \Downarrow \dots \Downarrow xs_n \setminus x_n$.

Note that our pattern specifications are closed formulae, containing no free variables. Although the names given to component variables greatly improve readability, they have no effect on semantics so, in the sequel, we will often omit new variable names and write simply $P \Downarrow xs$ to represent $P \Downarrow xs \setminus x$.

E. Generalisation operator

Definition 5: (Generalisation operator)

Let P be a pattern, $x : T \in Vars(P)$ and $xs \notin Vars(P)$. The *generalisation* of P on variable x , written $P \Uparrow x \setminus xs$, is defined as follows.

- 1) $Vars(P \Uparrow x \setminus xs) =$

$$(Vars(P) - \{x : T\}) \cup \{xs : \mathbb{P}(T)\},$$
- 2) $Pred(P \Uparrow x \setminus xs) = \forall x \in xs \cdot Pred(P).$ \square

For example, we can define the Adapter pattern as a generalisation of the variant $Adapter_1$, as follows:

$Adapter \triangleq$

$$(Adapter_1 \Uparrow request \setminus requests) \Uparrow specreq \setminus specreqs$$

We will use the same syntactic sugar for \Uparrow as we do for \Downarrow . We will often omit the new variable name and write $P \Uparrow x$. Thanks to an analogue of Equation 2, we can and will promote the operator \Uparrow to sets also.

F. Lift operator

The lift operator was first introduced in our previous work [44]. The definition given below is a revised version that allows lifting not only on class type variables but on variables of other types too.

Definition 6: (Lift Operator)

Let P be a pattern, $X = \{x_1 : T_1, \dots, x_k : T_k\} \subset Vars(P)$, $k > 0$ and $Pred(P) = p(x_1, \dots, x_n)$, where $n \geq k$. The lifting of P with X as the key, written $P \Uparrow X$, is the pattern defined as follows.

- 1) $Vars(P \uparrow X) = \{xs_1 : \mathbb{P}(T_1), \dots, xs_n : \mathbb{P}(T_n)\}$,
- 2) $Pred(P \uparrow X) = \forall x_1 \in xs_1 \dots \forall x_k \in xs_k \cdot$
 $\exists x_{k+1} \in xs_{k+1} \dots \exists x_n \in xs_n \cdot p(x_1, \dots, x_n)$. \square

When the key set is singleton, we omit the set brackets for simplicity, so we write $P \uparrow x$ instead of $P \uparrow \{x\}$.

Informally, lifting a pattern P results in a new pattern P' that contains a number of instances of pattern P . For example, *Adapter* \uparrow *Target* is the pattern that contains a number of *Targets* of adapted classes. Each of these has a dependent *Client*, *Adapter* and *Adaptee* class configured as in the original *Adapter* pattern. In other words, the component *Target* in the lifted pattern plays a role similar to the *primary key* in a relational database. Figure 3 is the pattern defined by expression *Adapter* \uparrow *Target*.

Specification 3: (Lifted Object Adapters Pattern) Components

- 1) Targets, Adapters, Adaptees, Clients \subseteq classes,
- 2) requestses, specreqses $\subseteq \mathbb{P}(\text{operations})$

Dynamic Components

- 1) mrs, mss \subseteq messages,

Conditions

- $$\forall Target \in Targets, \exists Client \in Clients,$$
- $$\exists Adapter \in Adapters, \exists Adaptee \in Adaptees,$$
- $$\exists requests \in requestses, \exists specreqs \in specreqses,$$
- $$\exists mr \in mrs, \exists ms \in mss.$$

1. Static Conditions

- 1) $requests \subseteq Target.ops$,
- 2) $specreqs \subseteq Adaptee.ops$
- 3) $Adapter \rightarrow Target$
- 4) $Adapter \rightarrow Adaptee$
- 5) $Client \rightarrow Target$

2. Dynamic Conditions

- 1) $mr.sig \in requests$
- 2) $ms.sig \in specreqs$
- 3) $trigs(mr, ms)$

Figure 3. Specification of Lifted Object Adapter Pattern

IV. EXAMPLES

In this section, we present two examples of using the operators to define composition of design patterns.

A. Model-View-Controller as Pattern Composition

Model-View-Controller (MVC) is one of the most well-known design patterns and perhaps the most widely used one. A detailed description of the MVC design pattern can be found in [47], which includes the class and sequence diagrams displayed in Figure 4. We can formalise the pattern as shown in Figure 5.

It is immediately apparent from the diagrams that the View and Controller classes are both observers of the Model, so we can alternatively specify MVC as an extension of the Observer pattern, whose specification is given in Figure 6.

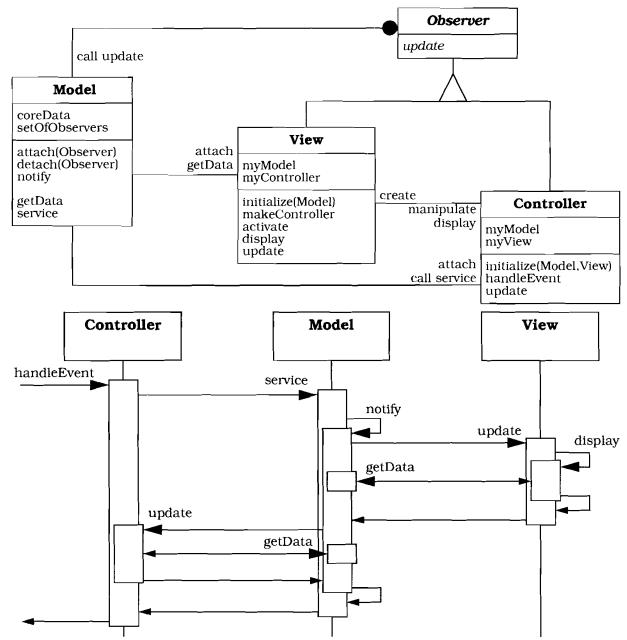


Figure 4. Class and Sequence Diagrams of the MVC Design Pattern

Note that the GoF book puts the *notify* operation in an abstract superclass, but we flatten the inheritance hierarchy for simplicity.

Now, we rename the variables in this pattern in two different ways to match those used for its two occurrences within MVC. We call these renamed patterns *Observer*₁ and *Observer*₂.

$$Observer_0 \triangleq$$

$$Observer[Model := Subject][getData := getState]$$

$$Observer_1 \triangleq$$

$$Observer_0[mu_1, mg_1 := mu, mg]$$

$$[View := ConcreteSubject]$$

$$Observer_2 \triangleq$$

$$Observer_0[mu_2, mg_2 := mu, mg]$$

$$[Controller := ConcreteSubject]$$

So, MVC pattern can now be defined as follows.

$$MVC \triangleq$$

$$(Observer_1 * Observer_2)$$

$$\#(\{display \in View.ops, mh, md \in messages,$$

$$handleEvent \in Controller.ops\})$$

$$\bullet(Observer \rightarrow View \wedge$$

$$mh.sig = handleEvent \wedge md.sig = display \wedge$$

$$trigs(mu_1, md) \wedge trigs(md, mg_1)$$

Here, $*$ is the operator that renames shared variable names

Specification 4: (MVC – Version 1)
Components
1) $Model, View, Controller, Observer \in \text{classes}$
2) $\text{notify}, \text{getData}, \text{service} \in \text{operations}$
3) $\text{display}, \text{handleEvent}, \text{update} \in \text{operations}$
Dynamic Components
1) $mh, ms, mn, mu_1, md, mg_1, mu_2, mg_2 \in \text{messages}$
Static Conditions
1) $\text{notify}, \text{getData}, \text{service} \in Model.\text{opers}$
2) $\text{display} \in View.\text{opers}$
3) $\text{handleEvent} \in Controller.\text{opers}$
4) $\text{update} \in Observer.\text{opers}$
5) $View \rightarrow Observer$
6) $Controller \rightarrow Observer$
7) $Model \rightarrow^* Observer$
8) $Controller \rightarrow View$
9) $Controller \rightarrow Model$
10) $View \rightarrow Model$
Dynamic Conditions
1) $mh.sig = handleEvent$
2) $ms.sig = service$
3) $mn.sig = notify$
4) $mu_1.sig = View.update$
5) $md.sig = display$
6) $mg_1.sig = getData$
7) $mu_2.sig = Controller.update$
8) $mg_2.sig = getData$
9) $\text{trigs}(mh, ms)$
10) $\text{trigs}(ms, mn)$
11) $\text{trigs}(mn, mu_1)$
12) $\text{trigs}(mu_1, md)$
13) $\text{trigs}(md, mg_1)$
14) $\text{trigs}(mu_1, mg_1)$
15) $\text{trigs}(mn, mu_2)$
16) $\text{trigs}(mu_2, mg_2)$

Figure 5. Specification of MVC Pattern (Version 1)

before applying * and then renames them back to what they were. Formally, let P_1 and P_2 be any given patterns, $\{v\} = Vars(P_1) \cap Vars(P_2)$ and $v_1 \neq v_2 \notin Vars(P_1) \cup Vars(P_2)$. Then, * is defined as follows, with the obvious generalisation to more than one variable:

$$P_1 * P_2 \triangleq$$

$$(P_1[v_1 := v] * P_2[v_2 := v])[v := v_1 = v_2].$$

The GoF book further proposes the use of Composite with MVC, to enable views to be nested, and Strategy too, so that the controller associated with each view is dynamically configurable. The specification of Strategy pattern is given in Figure 7. But, it is its lifted version composed with Composite, which is defined as follows.

$$\text{StrategyLifted} \triangleq$$

$$\text{Strategy} \uparrow \text{ConcreteStrategy} \setminus \text{ConcreteStrategies}$$

This brings us to a new definition of MVC, i.e., MVC_2 below. The result of evaluating this definition gives the specification shown in Figure 8.

Specification 5: (Simplified Observer)
Components
1) $Subject, ConcreteObserver, Observer \in \text{classes}$
2) $\text{notify}, \text{getState}, \text{service}, \text{update} \in \text{operations}$
Dynamic Components
1) $ms, mn, mu, mg \in \text{messages}$
Static Conditions
1) $\text{notify}, \text{getState}, \text{service} \in Subject.\text{opers}$
2) $\text{update} \in Observer.\text{opers}$
3) $ConcreteObserver \rightarrow Observer$
4) $Subject \rightarrow^* Observer$
5) $ConcreteObserver \rightarrow Subject$
Dynamic Conditions
1) $ms.sig = service$
2) $mn.sig = notify$
3) $mu.sig = ConcreteObserver.update$
4) $mg.sig = getState$
5) $\text{trigs}(ms, mn)$
6) $\text{trigs}(mn, mu)$
7) $\text{trigs}(mu, mg)$

Figure 6. Specification of Observer Pattern

Specification 6: (Strategy)
Components
1) $Context, Strategy, ConcreteStrategy \in \text{classes}$
2) $\text{contextInterface}, \text{algorithmInterface} \in \text{operations}$
Dynamic Components
1) $mc, ma \in \text{messages}$
Static Conditions
1) $\text{contextInterface} \in Context.\text{opers}$
2) $\text{algorithmInterface} \in Strategy.\text{opers}$
3) $Context \diamond \rightarrow Strategy$
4) $ConcreteStrategy \rightarrow Strategy$
5) $\text{algorithmInterface.isAbstract}$
6) $\neg \text{isAbstract}(ConcreteStrategy)$
Dynamic Conditions
1) $mc.sig = contextInterface$
2) $ma.sig = ConcreteStrategy.algorithmInterface$
3) $\text{trigs}(mc, ma)$

Figure 7. Specification of Strategy Pattern

$$\begin{aligned}
 MVC' &\triangleq \\
 MVC * (\text{Composite}[\text{display} = \text{operation} \wedge \\
 &\quad \text{View} = \text{Component} \wedge \text{Controller} = \text{Client}]) \\
 &\quad [\text{LeafView} := \text{Leaf}] \\
 &\quad [\text{CompositeView} := \text{Composite}] \\
 MVC_2 &\triangleq \\
 &(MVC' * \text{StrategyLifted}) \\
 &[\text{Controller} = \text{Strategy} \wedge \text{View} = \text{Context} \wedge \\
 &\quad \text{handleEvent} = \text{algorithmInterface} \wedge \\
 &\quad \text{actionPerformed} = \text{contextInterface} \wedge \\
 &\quad \text{ConcreteControllers} := \text{ConcreteStrategies}]
 \end{aligned}$$

Specification 7: (MVC – Version 2)
Components
1) $Model, View, Controller, Observer$ $Client, LeafView, CompositeView \in \text{classes}$
2) $notify, getData, service \in \text{operations}$
3) $display, handleEvent, update \in \text{operations}$
Dynamic Components
1) $mh, ms, mn, mu_1, md, mg_1, mu_2, mg_2,$ $m_1, m_2, mc, ma \in \text{messages}$
Static Conditions
1) $notify, getData, service \in Model.\text{opers}$
2) $display \in View.\text{opers}$
3) $handleEvent \in Controller.\text{opers}$
4) $update \in Observer.\text{opers}$
5) $View \rightarrow Observer$
6) $Controller \rightarrow Observer$
7) $Model \rightarrow^* Observer$
8) $Client \rightarrow View$
9) $Controller \rightarrow Model$
10) $View \rightarrow Model$
11) $LeafView \rightarrow View$
12) $CompositeView \rightarrow View$
13) $CompositeView \diamond \rightarrow^* View$
14) $\neg LeafView \diamond \rightarrow^* View$
15) $display.\text{isAbstract}$
16) $View \diamond \rightarrow Controller$
17) $\forall C \in ConcreteControllers \cdot C \rightarrow Controller$
18) $handleEvent.\text{isAbstract}$
19) $\forall C \in ConcreteControllers \cdot \neg \text{isAbstract}(C)$
Dynamic Conditions
1) $mh.sig = handleEvent$
2) $ms.sig = service$
3) $mn.sig = notify$
4) $mu_1.sig = View.update$
5) $md.sig = display$
6) $mg_1.sig = getData$
7) $mu_2.sig = Controller.update$
8) $mg_2.sig = getData$
9) $m_1.sig = Composite.operation$
10) $\text{isOp}(m_2)$
11) $mc.sig = contextInterface$
12) $ma.sig = ConcreteStrategy.algorithmInterface$
13) $\text{trigs}(mh, ms)$
14) $\text{trigs}(ms, mn)$
15) $\text{trigs}(mn, mu_1)$
16) $\text{trigs}(mu_1, md)$
17) $\text{trigs}(md, mg_1)$
18) $\text{trigs}(mu_1, mg_1)$
19) $\text{trigs}(mn, mu_2)$
20) $\text{trigs}(mu_2, mg_2)$
21) $\text{trigs}(mc, ma)$
22) $\text{trigs}(m_1, m_2)$
23) $m_2..sig = Leaf.operation \implies \neg \exists m_3 \in \text{messages} \cdot \text{trigs}(m_2, m_3) \wedge \text{isOp}(m_3)$

Figure 8. Specification of MVC Pattern (Version 2)

B. A Request-Handling Framework

In [32], the utility of pattern composition was demonstrated with a case study of pattern-based software design, in which five design patterns were composed to form an

extensible request-handling framework. As shown in Figure 9, the five patterns are Command, Command Processor, Memento, Strategy and Composite. The composition can be expressed in terms of our operators and an explicit definition of the pattern can thereby be derived.

The last two patterns have already been defined, thus here are the first three, starting with Command shown in Figure 10, which is based on the simplified version in [32] that makes the Client also be the invoker.

The original case study treats the memento as being created by the caretaker, but in fact it is created by the originator instead, so we have the specification of Memento in Figure 11.

The Command Processor pattern is not one of the GoF patterns. Figure 12 is the diagram given in [9] that illustrates the pattern's structure and dynamic behaviour. In particular, the Command Processor object executes requests on behalf of the clients. Its specification is given in Figure 13.

Now, the request-handling framework, $ReqHand$, can be defined as follows using our operators on patterns, where RH_1 , RH_2 and RH_3 are intermediate steps of the composition.

$$\begin{aligned}
RH_1 &\triangleq ((Command[Application := Receiver] \\
&\quad \uparrow ConcreteCommand \setminus ConcreteCommands \\
&\quad \uparrow mn \setminus mns \uparrow me \setminus mes) \\
&\quad * CommandProcessor[mee := me]) \\
&\quad [Component = Command] \\
RH_2 &\triangleq (RH_1 * Memento) \\
&\quad [Command \rightarrow Application \wedge \\
&\quad \quad Command = Caretaker \wedge \\
&\quad \quad Originator = Application] \\
RH_3 &\triangleq (RH_2 * Strategy \uparrow ma \setminus mas \\
&\quad \uparrow ConcreteStrategy \setminus ConcreteStrategies) \\
&\quad [CommandProcessor = Context] \\
&\quad [Strategy := Logging] \\
&\quad [ConcreteStrategies \\
&\quad \quad := ConcreteLoggingStrategies] \\
ReqHand &\triangleq \\
&\quad (RH_3 * Composite \uparrow m_2 \setminus mls) \\
&\quad \uparrow Leaf \setminus Leaves [Command = Component] \\
&\quad [mm := m] [LeafCommands := Leaves] \\
&\quad [ConcreteCommands = \\
&\quad \quad LeafCommands \cup \{CompositeCommands\}] \\
&\quad [CompositeCommand := Composite]
\end{aligned}$$

Evaluating the above expressions according to the definitions of the operators, we have the specification of the extensible request handling framework shown in Figure 14 for the static and dynamic parts.

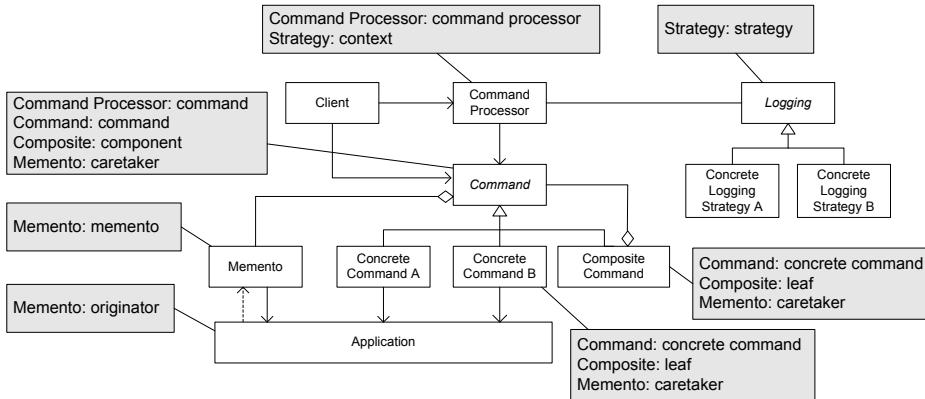


Figure 9. Request Handling Framework

Specification 8: (Command) Components

- 1) $\text{Command} \in \text{classes}$
- 2) $\text{ConcreteCommand} \in \text{classes}$
- 3) $\text{Client} \in \text{classes}$
- 4) $\text{Receiver} \in \text{classes}$
- 5) $\text{execute}, \text{action} \in \text{operations}$

Dynamic Components

- 1) $m_n, m_e, m_a \in \text{messages}$

Static Conditions

- 1) $\text{execute} \in \text{Command.ops}$
- 2) $\text{action} \in \text{Receiver.ops}$
- 3) $\text{Client} \rightarrow \text{Command}$
- 4) $\text{ConcreteCommand} \rightarrow \text{Receiver}$
- 5) $\text{ConcreteCommand} \rightarrow \text{Command}$
- 6) $\text{execute.isAbstract}$
- 7) $\neg\text{isAbstract}(\text{ConcreteCommand})$

Dynamic Conditions

- 1) $m_n.sig.isNew$
- 2) $m_e.sig = \text{execute}$
- 3) $m_a.sig = \text{action}$
- 4) $m_n < m_e$
- 5) $\text{fromLL}(m_n).class = \text{Client}$
- 6) $\text{fromLL}(m_e).class = \text{Client}$
- 7) $\text{toLL}(m_n) = \text{toLL}(m_e)$
- 8) $\text{trigs}(m_e, m_a)$

Figure 10. Specification of Command Pattern

Specification 9: (Memento) Components

- 1) $\text{Caretaker}, \text{Memento}, \text{Originator} \in \text{classes}$
- 2) $\text{setState}, \text{getState} \in \text{operations}$
- 3) $\text{createMemento}, \text{setMemento} \in \text{operations}$

Dynamic Components

- 1) $mcm, mn, mss, msm, mgs \in \text{messages}$

Static Conditions

- 1) $\text{setState}, \text{getState} \in \text{Memento.ops}$
- 2) $\text{createMemento}, \text{setMemento} \in \text{Originator.ops}$
- 3) $\text{Caretaker} \diamond \rightarrow \text{Memento}$

Dynamic Conditions

- 1) $mcm.sig = \text{createMemento}$
- 2) $mn.sig.isNew$
- 3) $mss.sig = \text{setState}$
- 4) $msm.sig = \text{setMemento}$
- 5) $mgs.sig = \text{getState}$
- 6) $\text{trigs}(mcm, mn)$
- 7) $\text{trigs}(mcm, mss)$
- 8) $\text{trigs}(mss, mgs)$
- 9) $mcm < msm$
- 10) $\text{fromLL}(mcm) = \text{fromLL}(msm)$
- 11) $\text{toLL}(mcm) = \text{toLL}(msm)$
- 12) $\text{hasParam}(msm, \text{toLL}(gs))$
- 13) $\text{toLL}(mn) = \text{returnValue}(mn)$
- 14) $\text{toLL}(mss) = \text{returnValue}(mss)$

Figure 11. Specification of Memento Pattern

Specification 10: (Command Processor) Components

- 1) $\text{Client}, \text{CommandProcessor}, \text{Component} \in \text{classes}$
- 2) $\text{executeRequest}, \text{function} \in \text{operations}$
- 3) $m_e, mf \in \text{messages}$

Static Conditions

- 1) $\text{executeRequest} \in \text{CommandProcessor.ops}$
- 2) $\text{function} \in \text{Component.ops}$
- 3) $\text{Client} \rightarrow \text{CommandProcessor}$
- 4) $\text{CommandProcessor} \rightarrow \text{Component}$

Dynamic Conditions

- 1) $m_e.sig = \text{executeRequest}$
- 2) $mf.sig = \text{function}$
- 3) $\text{fromLL}(m_e).class = \text{Client}$
- 4) $\text{trigs}(m_e, mf)$

Figure 13. Specification of Command Processor Pattern

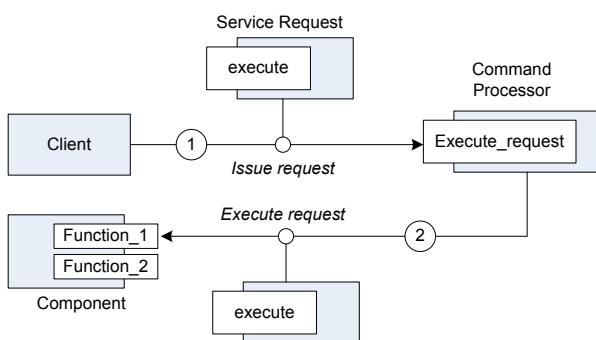


Figure 12. Diagram of Command Processor Pattern [9]

<p>Specification 11: (Extensible Request Handler) Components</p> <ol style="list-style-type: none"> 1) $Command, Client, Application, CommandProcessor, Memento, Logging, Client, CompositeCommand \in \text{classes}$ 2) $ConcreteCommands, LeafCommands, ConcreteLoggingStrategies \subseteq \text{classes}$ 3) $\text{execute}, \text{function}, \text{operation}, \text{action} \in \text{operations}$ 4) $\text{executeRequest}, \text{contextInterface} \in \text{operations}$ 5) $\text{setState}, \text{getState} \in \text{operations}$ 6) $\text{createMemento}, \text{setMemento}, \text{algorithmInterface} \in \text{operations}$ <p>Static Conditions</p> <ol style="list-style-type: none"> 1) $\text{execute}, \text{function}, \text{operation} \in \text{Command.ops}$ 2) $\text{action} \in \text{Action.ops}$ 3) $\text{executeRequest}, \text{contextInterface} \in \text{CommandProcessor.ops}$ 4) $\text{setState}, \text{getState} \in \text{Memento.ops}$ 5) $\text{createMemento}, \text{setMemento} \in \text{Application.ops}$ 6) $\text{algorithmInterface} \in \text{Logging.ops}$ 7) $\text{Client} \rightarrow \text{Command}$ 8) $\forall C \in \text{ConcreteCommands} : C \rightarrow \text{Application}$ 9) $\forall C \in \text{ConcreteCommands} : C \rightarrow \text{Command}$ 10) $\text{execute.isAbstract}$ 11) $\forall C \in \text{ConcreteCommands} : \neg \text{isAbstract}(C)$ 12) $\text{Client} \rightarrow \text{CommandProcessor}$ 13) $\text{CommandProcessor} \rightarrow \text{Application}$ 14) $\text{Caretaker} \diamond \rightarrow \text{Memento}$ 15) $\text{Command} \rightarrow \text{Application}$ 16) $\text{Memento} \rightarrow \text{Application}$ 17) $\text{CommandProcessor} \diamond \rightarrow \text{Logging}$ 18) $\forall C \in \text{ConcreteLoggingStrategies} : C \rightarrow \text{Logging}$ 19) $\text{algorithmInterface.isAbstract}$ 20) $\forall C \in \text{ConcreteLoggingStrategies} : \neg \text{isAbstract}(C)$ 21) $\text{CompositeCommand} \rightarrow \text{Command}$ 22) $\text{CompositeCommand} \diamond \rightarrow^* \text{Command}$ 23) $\forall C \in \text{LeafCommands} : \neg C \diamond \rightarrow^* \text{Command}$ 24) $\text{ConcreteCommands} = \text{LeafCommands} \cup \{\text{CompositeCommand}\}$ 25) $\text{operation.isAbstract}$ 	<p>Dynamic Components</p> <ol style="list-style-type: none"> 1) ma, mee, mf, mc, mm $mcm, mnm, mss, msm, mgs \in \text{messages}$ 2) $mns, mes, mas, mls \subseteq \text{messages}$ <p>Dynamic Conditions</p> <ol style="list-style-type: none"> 1) $\forall C \in \text{ConcreteCommands} : mns_C.sig.isNew$ 2) $\forall C \in \text{ConcreteCommands} : msc.sig = C.execute$ 3) $ma.sig = \text{action}$ 4) $mee.sig = \text{executeRequest}$ 5) $mf.sig = \text{function}$ 6) $mc.sig = \text{createMemento}$ 7) $mnm.sig.isNew$ 8) $mss.sig = \text{setState}$ 9) $msm.sig = \text{setMemento}$ 10) $mgs.sig = \text{getState}$ 11) $mc.sig = \text{contextInterface}$ 12) $\forall C \in \text{ConcreteStrategies} : msc.sig = C.algorithmInterface$ 13) $mm.sig = \text{Composite.operation}$ 14) $\forall C \in \text{LeafCommands} : \text{isOp}(mls_C)$ 15) $\forall C : mns_C < msc$ 16) $\forall C : \text{fromLL}(mns_C).class = \text{Client}$ 17) $\forall C : \text{fromLL}(msc).class = \text{Client}$ 18) $\forall C : \text{toLL}(mns_C) = \text{toLL}(msc)$ 19) $\forall C : \text{trigs}(msc, ma)$ 20) $\text{fromLL}(mee).class = \text{Client}$ 21) $\text{trigs}(mee, mf)$ 22) $\text{trigs}(mcm, mnm)$ 23) $\text{trigs}(mcm, mss)$ 24) $\text{trigs}(mss, mgs)$ 25) $mcm < msm$ 26) $\text{fromLL}(mcm) = \text{fromLL}(msm)$ 27) $\text{toLL}(mcm) = \text{toLL}(msm)$ 28) $\text{hasParam}(msm, \text{toLL}(gs))$ 29) $\text{toLL}(mnm) = \text{returnValue}(mnm)$ 30) $\text{toLL}(mss) = \text{returnValue}(mnm)$ 31) $\forall C \in \text{ConcreteStrategies} : \text{trigs}(mc, msc)$ 32) $\forall C \in \text{LeafCommands} : \text{trigs}(mm, mls_C)$ 33) $\forall C \in \text{LeafCommands} : mls_C.sig = C.operation \Rightarrow \neg \exists mmm \in \text{messages} : \text{trigs}(mls_C, mmm) \wedge \text{isOp}(mmm)$
---	--

Figure 14. Specification of Request Handling Pattern

V. CASE STUDY

In the GoF book, the documentation for each pattern concludes with a brief section entitled Related Patterns. A few words are devoted to the comparisons and contrasts that this title would suggest, but the section mostly consists of suggestions for how other patterns may be composed with the one under discussion. These compositions are the subject of our case study.

On page 106 of the GoF book, for example, it is stated that *A Composite is what the builder often builds*. This suggests a composition of the Composite and Builder patterns, and that composition can formally be specified using our operators

as follows:

$$(Builder * Composite)[Product = Component].$$

Figure 15 shows the relationships between patterns that we have successfully formalised. The formal definitions of the relationships are given in Table II; the two numbers in each row are the arrow label followed by the page number in the GoF book. The column "Description of the Relationship" quotes what are described in the GoF book. The column "Formal Expression" gives the expression of the relationship using the operators.

A similar diagram appears in the GoF book but we have added five new arrows, numbered in bold font, for the

Table II
FORMAL DEFINITIONS OF THE COMPOSITIONAL RELATIONSHIPS BETWEEN PATTERNS

No.	Page	Description of the relationship	Formal expression
1	106	A Composite is what the builder often builds.	(Builder * Composite) [Product = Component]
2	173,	Often the component-parent link is used for a Chain of Responsibility.	(Composite * ChainOfResponsibility) [Handler = Component \wedge Operation = Handle \wedge multiplicity = 1]
232		Chain of Responsibility is often applied in conjunction with Composite. There, a component's parent can act as its successor.	
3	173	When Decorator and Composite are used together, they will usually have a common parent class.	(Composite' * Decorator) [Decorator = Composite' \wedge Composite'.Component = Decorator.Component \wedge Composite.Operation = Decorator.Operation \wedge ConcreteComponent = Leaf]
4	173	Flyweight lets you share components [of Composite].	(Composite * Flyweight)
206		The Flyweight pattern is often combined with the Composite pattern to implement a logically hierarchical structure in terms of a directed-acyclic graph with shared leaf nodes.	[Leafs = {ConcreteFlyweight, UnsharedConcreteFlyweight}]
5	173	Iterator can be used to traverse composites.	(Composite * Iterator') [ConcreteAggregate = Component]
6	173	Visitor localises operations and behaviour that would otherwise be distributed across composite and leaf classes [in the Composite].	(Composite * Visitor) [Element = Component \wedge Operation = Accept(v) \wedge ConcreteElements = {Leaf, Composite}]
7	242	A Composite can be used to implement MacroCommands [i.e., ConcreteCommand in Command].	(Composite * Command) [Command = Component \wedge execute = operation \wedge ConcreteCommand = Leaf]
8	255	Flyweight shows how to share terminal symbols within the abstract syntax tree.	(Interpreter * Flyweight) [TerminalExpression = Flyweight]
9	255	Visitor can be used to maintain the behaviour in each node in the abstract syntax tree in one class.	(Interpreter * Visitor) [Element = AbstractExpression \wedge Interpret = Accept(v) \wedge ConcreteElements = {NonTerminalExpression, TerminalExpression}]
10	95	AbstractFactory classes are often implemented with factory methods of Factory Method.	(AbstractFactory * ((FactoryMethod ↑ Product) $\uparrow\downarrow$ FactoryMethod)) [Creator = AbstractFactory \wedge #AnOperations = 1 \wedge Products = AbstractProducts \wedge createMethods \subseteq FactoryMethods \wedge ConcreteCreators = ConcreteFactories \wedge AbstractFactories \subseteq FactoryProducts = FactoryMethod.ConcreteProducts]
11	95	AbstractFactory classes can also be implemented using Prototype.	(AbstractFactory * (Prototype ↑ Client)) [ConcreteFactories \subseteq Clients \wedge AbstractProducts \subseteq Prototypes \wedge CreatePrototypes = ConcreteProducts]
12	95	A concrete factory in the AbstractFactory is often a singleton.	(AbstractFactory * (Singleton ↑ {Singleton})) [Singletons \subseteq ConcreteFactories]
13	116	Factory methods are often called within Template Methods.	(TemplateMethod * FactoryMethod)
14	193	Abstract Factory can be used with Facade to provide an interface for creating subsystem objects in a subsystem-independent way.	[AbstractClass = Creator \wedge TemplateMethod = AnOperation]
15	161	Abstract Factory can create and configure a particular bridge.	(AbstractFactory * Bridge) [AbstractProducts = {Abstraction, Implementor}]
16	193	usually only one Facade object is required. Thus Facade objects are often Singletons.	(Facade * Singleton)[Facade = Singleton]
17	242	A Memento can keep the state the command [in Command] requires to undo its effect.	(Command * Memento)[Originator = Command]
18	242	A command [in Command] that must be copied before being placed on the history list, acts as a Prototype.	(Command * Prototype)[Command = Prototype]
19	271	Polymorphic Iterators reply on factory methods to instantiate the appropriate Iterator subclass.	(Iterator*FactoryMethod) [ConcreteCreator = Aggregate \wedge Product = Iterator \wedge ConcreteProduct = ConcreteIterator \wedge AnOperation = CreateIterator]
20	271	An iterator can use a memento to capture the state of an iteration.	(Memento * Iterator) [ConcreteAggregate = Originator]
		The iterator stores the memento internally.	
21	282	Colleagues can communicate with the mediator using the Observer.	(Mediator * Observer) [ConcreteColleagues = {ConcreteSubject, ConcreteObserver}]
22	303	The ChangeManager [an instance of the Mediator pattern] may use the Singleton pattern to make it unique and globally accessible.	(Mediator * Singleton) [ConcreteMediator = Singleton]
23	313	The Flyweight pattern explains when and how State objects can be shared.	(Flyweight * State) [Flyweight = State \wedge Handle = Operation(extrinsicState)]
24	313	State objects are often Singletons.	(State * (Singleton $\uparrow\downarrow$ Singleton)) [Singletons \subseteq ConcreteStates]
25	206	It's often best to implement Strategy objects as Flyweight.	(Strategy * Flyweight)
			[Strategy = Flyweight \wedge algorithmInterface = Operation(extrinsicState)]

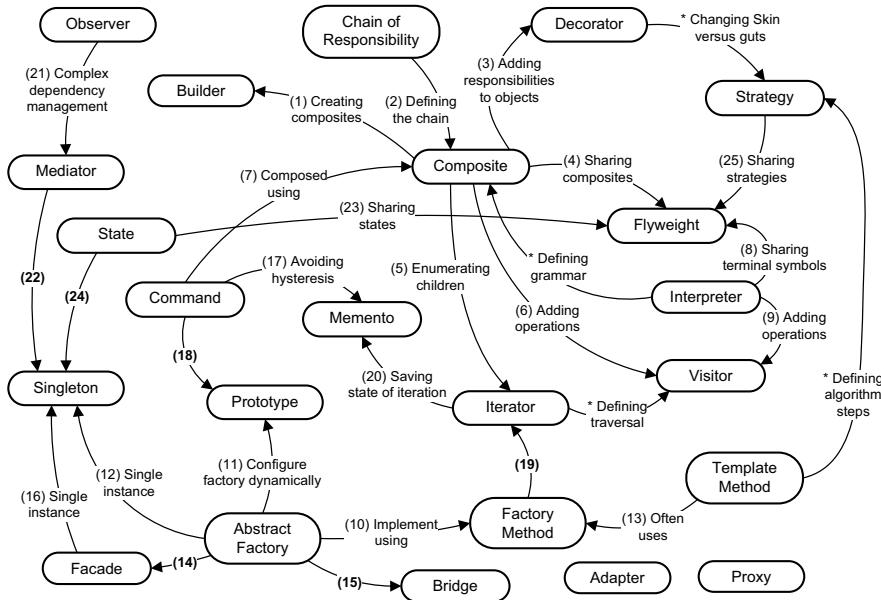


Figure 15. Case Study on Formalising Relationships between GoF Patterns

relationships we have formalised that are discussed in the main text but not shown on the original diagram. Four other relationships are unnumbered but asterisked. These do not represent compositions and so have not been formalised. In particular, and for a start, it is a specialisation relation that links Composite and Interpreter. The relationship between Decorator and Strategy is about the differences between them, not a suggested composition. So too is the relationship between Strategy and Template Method. And finally, the relationship between Iterator and Visitor, has been left unformalised for the different reason that it is mentioned in GoF only on the diagram, and not expanded upon in the main text. Therefore, our case study has covered all the compositional relationships in the GoF book.

Comparing Table II with Table 2 of [44], which express the same relationships using composition with overlaps, we can see that those compositional relationships that require one-to-many and many-to-many overlaps can all be represented more accurately using our operators.

In summary, the case studies demonstrated that the operators defined in this paper are expressive enough to define compositions of design patterns. Other work by us [44] has shown that their logic properties and algebraic laws are useful for proving the properties of pattern compositions.

VI. CONCLUSION

In this paper, we proposed a set of operators on design patterns that enable compositions to be formally defined with flexibility. We illustrated the operators with examples. We also reported a case study on the relationships suggested by

the GoF book [2]. This demonstrated the expressiveness of the operators when used to compose patterns.

A. Related Work

As far as we know, there is no similar work in the literature that defines operators on design patterns for pattern composition or instantiation. The closest work is perhaps that of Dong *et al.* [37] and Taibi [18], [42], [43], as previously discussed in Section I. Here we discuss the relationship between their work and ours more formally, using their notation for expositional clarity.

In [38], Dong *et al.* describe a composition P of patterns P_1, P_2, \dots, P_n using a *composition mapping* $C : P_1 \times \dots \times P_n \rightarrow P$. This is, in fact, intended to formally represent a set of signature mappings C_i such that C_i maps the sets of component names in pattern P_i to P so the properties θ_i for each P_i is translated into another property $\theta'_i = C(\theta_i)$ as a part of the properties of P . In [39], the composition mapping is better defined as from the union of the variables in P_i . For instantiation, the mapping is to constants of classes, attributes, methods, etc.

The approach of Taibi *et al.* [42], [43] is very similar except that they directly rename the components using substitution. Again, composition replaces variables with variables, whereas instantiation replaces them with constants. Formally, if pattern P_1 have properties φ_1 and pattern P_2 have properties φ_2 then the properties of their composition are given by

$$\text{Subst}\{v_1 \setminus t_1, \dots, v_n \setminus t_n\}, \varphi_1 \wedge \varphi_2,$$

which, informally, is the conjunction of φ_1 and φ_2 after variables v_i have each been replaced by terms t_i . Here,

terms t_i are either variables or constants. This approach has an advantage over that of Dong *et al.* that instantiation and composition are represented in the same notation, but apart from that it is mathematically equivalent, because substitutions are mappings with the terms restricted to be either variables or constants. Since substitutions and signature mappings must both preserve variable types for the translations to be syntactically valid, neither approach can express one-to-many or many-to-many overlaps. Moreover, they are both mathematically equivalent to an application of our restriction operator with conditions in the simplest form, $u = v$. That is why our approach is more expressive, as we have demonstrated in the case study.

B. Further work

Formal reasoning about design patterns and their compositions can naturally be supported by formal deduction in first-order logic. This activity is well understood, and well supported by software tools such as theorem provers. It is desirable to employ or develop such tools for automatic reasoning about pattern compositions that are expressed as applications of the operators.

We have seen that pattern compositions can be represented by different but equivalent expressions. For example, we saw in Section III that $Adapter_1$ can be expressed either using the restriction operator or by using the flatten operator, and these two expressions are equivalent. Inspired by this, we have investigated the algebraic laws that the operators obey. This led us to a calculus of pattern composition for reasoning about the equivalence of such expressions. The results have been reported in a separate paper [48], thus omitted here.

One of the more important questions in the study of pattern composition is whether a composition is appropriate for a particular pair of patterns. Dong *et al.* addressed this issue in [37] with their notion of faithfulness conditions. A composition is faithful to the composed patterns if it satisfies two conditions: (a) no pattern loses any properties after composition, and (b) the composition does not add any new facts to its components. However, Taibi and Ngo argued that although the first condition is relevant, it is not always necessary [43]. So further investigation seemed warranted on how to formalise the notion of appropriateness, and to prove that the operators presented in this paper have such a property.

REFERENCES

- [1] I. Bayley and H. Zhu, "A formal language of pattern composition," in *Proceedings of The 2nd International Conference on Pervasive Patterns (PATTERNS 2010)*. Xpert Publishing Services, Nov. 2010, pp. 1–6.
- [2] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns - Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.
- [3] D. Alur, J. Crupi, and D. Malks, *Core J2EE Patterns: Best Practices and Design Strategies*, 2nd ed. Prentice Hall, 2003.
- [4] M. Grand, *Patterns in Java: A Catalog of Reusable Design Patterns Illustrated with UML, Volume 1*. John Wiley & Sons, 2002.
- [5] ———, *Patterns in Java, volume 2*. John Wiley & Sons, 1999.
- [6] ———, *Java Enterprise Design Patterns*. John Wiley & Sons, 2002.
- [7] M. Fowler, *Patterns of Enterprise Application Architecture*. Addison Wesley, 2003.
- [8] G. Hohpe and B. Woolf, *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison Wesley, 2004.
- [9] F. Buschmann, K. Henney, and D. C. Schmidt, *Pattern-Oriented Software Architecture. vol. 4: A Pattern Language for Distributed Computing*. John Wiley & Sons, 2007.
- [10] M. Voelter, M. Kircher, and U. Zdun, *Remoting Patterns*. John Wiley & Sons, 2004.
- [11] M. Schumacher, E. Fernandez, D. Hybertson, and F. Buschmann, *Security Patterns: Integrating Security and Systems Engineering*. John Wiley & Sons, 2005.
- [12] C. Steel, *Applied J2EE Security Patterns: Architectural Patterns & Best Practices*. Prentice Hall, 2005.
- [13] L. DiPippo and C. D. Gill, *Design Patterns for Distributed Real-Time Systems*. Springer-Verlag, 2005.
- [14] B. P. Douglass, *Real Time Design Patterns: Robust Scalable Architecture for Real-time Systems*. Addison Wesley, 2002.
- [15] R. S. Hanmer, *Patterns for Fault Tolerant Software*. Wiley, 2007.
- [16] P. S. C. Alencar, D. D. Cowan, and C. J. P. de Lucena, "A formal approach to architectural design patterns," in *Proc. of FME'96*, Springer-Verlag, 1996, pp. 576 – 594.
- [17] T. Mikkonen, "Formalizing design patterns," in *Proc. of ICSE 1998*. IEEE CS, April 1998, pp. 115–124.
- [18] T. Taibi, D. Check, and L. Ngo, "Formal specification of design patterns-a balanced approach," *Journal of Object Technology*, vol. 2, no. 4, Jul.-Aug. 2003.
- [19] E. Gasparis, A. H. Eden, J. Nicholson, and R. Kazman, "The design navigator: charting Java programs," in *Proc. of ICSE'08*, Companion Volume, 2008, pp. 945–946.
- [20] I. Bayley and H. Zhu, "Formal specification of the variants and behavioural features of design patterns," *Journal of Systems and Software*, vol. 83, no. 2, pp. 209–221, Feb. 2010.
- [21] J. Niere, W. Schäfer, J. P. Wadsack, L. Wendehals, and J. Welsh, "Towards pattern-based design recovery," in *Proc. of ICSE 2002*. IEEE CS Press, May 2002, pp. 338–348.
- [22] D. Hou and H. J. Hoover, "Using SCL to specify and check design intent in source code," *IEEE Transactions on Software Engineering*, vol. 32, no. 6, pp. 404–423, June 2006.
- [23] N. Nija Shi and R. Olsson, "Reverse engineering of design patterns from Java source code," in *Proc. of ASE 2006*, Sept. 2006, pp. 123–134.
- [24] A. Blewitt, A. Bundy, and I. Stark, "Automatic verification of design patterns in Java," in *Proc. of ASE 2005*. ACM Press, Nov. 2005, pp. 224–232.
- [25] D. Mapelsden, J. Hosking, and J. Grundy, "Design pattern modelling and instantiation using dpml," in *Proc. of Tools Pacific 2002*. Australian Computer Society, 2002, pp. 3–11.
- [26] J. Dong, Y. Zhao, and T. Peng, "Architecture and design pattern discovery techniques - a review," in *Proc. of SERP 2007*, H. R. Arabnia and H. Reza, Eds., vol. II. CSREA Press, Jun. 25-28 2007, pp. 621–627.
- [27] D.-K. Kim and L. Lu, "Inference of design pattern instances in UML models via logic programming," in *Proc. of ICECCS 2006*. IEEE CS Press, Aug. 2006, pp. 47–56.

- [28] D.-K. Kim and W. Shen, "An approach to evaluating structural pattern conformance of UML models," in *Proc. of SAC'07*. ACM Press, March 2007, pp. 1404–1408.
- [29] —, "Evaluating pattern conformance of UML models: a divide-and-conquer approach and case studies," *Software Quality Journal*, vol. 16, no. 3, pp. 329–359, 2008.
- [30] H. Zhu, I. Bayley, L. Shan, and R. Amphlett, "Tool support for design pattern recognition at model level," in *Proc. of COMPSAC 2009*. IEEE CS Press, Jul. 2009, pp. 228–233.
- [31] H. Zhu, L. Shan, I. Bayley, and R. Amphlett, "A formal descriptive semantics of UML and its applications," in *UML 2 Semantics and Applications*, K. Lano, Ed. John Wiley & Sons, Nov. 2009.
- [32] F. Buschmann, K. Henney, and D. C. Schmidt, *Pattern-Oriented Software Architecture. vol. 5: On Patterns and Pattern Languages*. John Wiley & Sons, 2007.
- [33] D. Riehle, "Composite design patterns," in *Proc. of OOPSLA'97*. ACM Press, Oct. 1997, pp. 218–228.
- [34] J. Vlissides, "Notation, notation, notation," *C++ Report*, Apr. 1998.
- [35] J. Dong, S. Yang, and K. Zhang, "Visualizing design patterns in their applications and compositions," *IEEE Transactions on Software Engineering*, vol. 33, no. 7, pp. 433–453, Jul. 2007.
- [36] J. M. Smith, "The pattern instance notation: A simple hierarchical visual notation for the dynamic visualization and comprehension of software patterns," *Journal of Visual Languages and Computing*, vol. 22, no. 5, pp. 355–374, Oct. 2011, doi:10.1016/j.jvlc.2011.03.003.
- [37] J. Dong, P. S. Alencar, and D. D. Cowan, "Ensuring structure and behavior correctness in design composition," in *Proc. of ECBS 2000*. IEEE CS Press, Apr. 2000, pp. 279–287.
- [38] J. Dong, P. S. C. Alencar, and D. D. Cowan, "Correct composition of design components," in *Proc. of the 4th International Workshop on Component-Oriented Programming in conjunction with ECOOP'99*, 1999.
- [39] J. Dong, P. S.C.Alencar, and D. Cowan, "A behavioral analysis and verification approach to pattern-based design composition," *Software and Systems Modeling*, vol. 3, pp. 262–272, 2004.
- [40] J. Dong, T. Peng, and Y. Zhao, "Automated verification of security pattern compositions," *Information and Software Technology*, vol. 52, no. 3, p. 274–295, Mar. 2010.
- [41] —, "On instantiation and integration commutability of design pattern," *The Computer Journal*, vol. 54, no. 1, pp. 164–184, Jan. 2011.
- [42] T. Taibi, "Formalising design patterns composition," *Software, IEE Proceedings*, vol. 153, no. 3, pp. 126–153, Jun. 2006.
- [43] T. Taibi and D. C. L. Ngo, "Formal specification of design pattern combination using BPSL," *Information and Software Technology*, vol. 45, no. 3, pp. 157–170, March 2003.
- [44] I. Bayley and H. Zhu, "On the composition of design patterns," in *Proc. of QSIC 2008*. IEEE CS Press, Aug. 2008, pp. 27–36.
- [45] H. Zhu, "On the theoretical foundation of meta-modelling in graphically extended BNF and first order logic," in *Proc. TASE 2010*. IEEE CS Press, Aug. 2010, pp. 95–104.
- [46] —, "An institution theory of formal meta-modelling in graphically extended BNF," *Frontier of Computer Science*, (In Press).
- [47] F. Buschmann, K. Henney, and D. C. Schmidt, *Pattern-oriented Software Architecture. Vol. 1: A System of Patterns*. John Wiley & Sons, 1996.
- [48] H. Zhu and I. Bayley, "Laws of pattern composition," in *Proc. of ICFEM 2010*, LNCS, vol. 6447. Springer, Nov. 17-19 2010, pp. 630–645.

A Pattern-based Adaptation for Abstract Applications in Pervasive Environments

Imen Ben Lahmar*, Djamel Belaïd* and Hamid Mukhtar†

*Institut Telecom; Telecom SudParis, CNRS UMR SAMOVAR, Evry, France

Email: {imen.ben_lahmar, djamel.belaïd}@it-sudparis.eu

†National University of Sciences and Technology, Islamabad, Pakistan

Email: hamid.mukhtar@seecs.edu.pk

Abstract—Using service-oriented architecture, applications can be defined as an assembly of abstract components that are mapped to a concrete level to fulfill their executions. However, several problems may be detected during their mapping as well as during their executions, which prevent them to be executed successfully. Thus, there is a need to adapt them according to the given contexts. In this article, we present some situational contexts that may trigger the adaptation of applications at init time or during their execution. Upon detection of certain changes in context, the applications are adapted accordingly. For this goal, we propose a set of adaptation patterns that provide an extra-functional behavior with respect to the functional behavior of the applications. These patterns are injected into abstract applications if a relevant context is sensed to ensure their mapping as well as their execution.

Keywords-Adaptation patterns, mismatches, abstract applications, component model.

I. INTRODUCTION

The proliferation of small devices and the increase in number of services created by various vendors for such devices have made Service-Oriented Architecture (SOA) a primary choice for mobile software developers. One particular approach for developing SOA-based applications is to use component-based application development.

Using this approach, an application is defined as an assembly of loosely-coupled components, requiring services from and providing services to each other. Complex applications can be crafted using an arbitrarily large number of software components. Specifically, when developing business applications using SOA, it becomes inevitable to implement the business functionality by a mix of self-contained, reusable and loosely connected components.

In such approach, it is possible to represent an application by an assembly of abstract components (i.e., services), which leads to automatic selection of services across various devices in the environment. At the time of execution, the services are mapped to concrete components, distributed across various devices.

To illustrate our point of view, let's consider a video player application that provides the functionality of displaying video to the user. The user is also able to control the playback of the application. The application is represented by an assembly of abstract components, which describe only the services required or provided by the application namely, controlling, decoding

and displaying video. The application has to be mapped to the concrete components to achieve its realization.

The complexities involved in designing and realizing such applications have been identified and addressed by many previous approaches [2] [6] [15].

While the existing approaches may assume that a mapping from abstract to concrete application can be done effortlessly, many problems may arise at init time that prevent it to be achieved successfully for example the heterogeneity of interfaces of connections between devices. Furthermore, applications in pervasive environments are challenged by the dynamism of their execution environment due to, e.g., user and device mobility, which make them subjects to unforeseen failures.

These problems imply mismatches between the abstract application and the concrete level occurring at init time or during the execution of the application. That is, the application cannot be executed in the given context or in the new context if it changes. Therefore, applications have to be adapted in order to carry out their mapping, and subsequently, their execution.

In literature, we distinguish two main adaptive techniques, namely parametric and compositional mechanisms to adapt applications in pervasive environment [14]. The parametrization techniques aim at adjusting internal or global parameters, while the compositional adaptations allow the replacement of components implementations or the modification of the applications structure.

In our work, we are interested in the last category, i.e., the structural adaptation, to overcome the mismatches between the abstract application and the concrete level with respect to the functional behavior of the application.

Therefore, in our previous work [4], we have proposed to transform an abstract application to another one by injecting adaptation patterns into the abstract application, which provide an extra-functional behavior allowing the mapping and the execution of the application. To facilitate the description of the adaptation patterns, we have defined a generic adapter template that encapsulates the main features of an adapter.

In this paper, we make the following novel contributions, some of which extend our previous work [4]: 1) we identify some situational contexts according to which the application can be adapted and 2) we propose a set of adaptation patterns that define the adaptation behavior of an application given a

certain context.

The rest of the article has been organized as follows. Section II presents the adaptation context that may trigger the adaptation of abstract applications. Section III describes the principle of our structural adaptation approach and the proposed set of adaptation patterns. In Section IV, we present an example scenario through which we give an architectural description of applications and patterns. In Section V, we present some implementation details, whereas, in Section VI, we provide an overview of existing related approaches as well as their limitations. Finally, Section VII concludes this article with some future directions.

II. ADAPTATION CONTEXTS

A. Classification of Mismatches

A generalized notion of context has been proposed in [1] as *any information that can be used to characterize the situation of an entity (person, location, object, etc.).* We consider adaptation context as any piece of information that may trigger the adaptation of the application.

We are interested in contexts that represent the mismatches between the abstract descriptions of applications and the concrete level. These mismatches imply that the current abstract description could not be realized in the given context, or in the new context, if it has changed.

We have classified the mismatches level, where they occur, into three categories: inter-components, intra-device and intra-devices mismatches (see Figure 1). This categorization covers not only the software mismatches but also the network and hardware problems.

At the inter-components level, we consider the mismatches that may arise at init time due to the non-satisfaction of the non-functional requirements of the components. These latter are system requirements which are not of a functional nature, but contribute decisively to the applicability of the system [9] like security, reliability, performance, etc. Thus, if they are not ensured at the concrete level, this may prevent the abstract application to be well mapped.

At this level, the mismatches may be also related to the heterogeneity of signatures, protocols or semantic [3]. In the present work, we do not focus on these mismatches as it has been previously studied [12] [13] [19]. However, it is possible to resolve them using our approach.

It is also possible to detect mismatches at intra-device level. These mismatches denote that the desired characteristics of devices, as specified by a service or a user, are not satisfied due to their reduced capacities like using a lower battery power, a slower CPU, etc. Thus, there is a need to adapt the abstract application to consider these requirements.

In case of a distributed environment, there is also a need to consider the mismatches occurring at inter-devices level. These mismatches may be related to the use of heterogeneous interfaces of connection, a lower bandwidth, etc. Thus, they have an impact in the communication between devices.

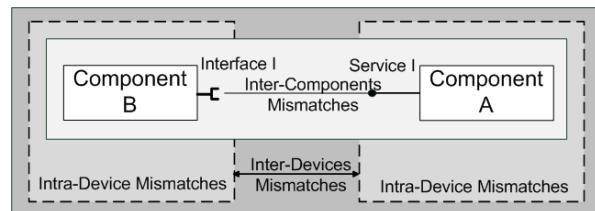


Fig. 1. Categorization of Mismatches levels

B. Detection of Mismatches

The application resolution and execution is ensured by our Middleware for Monitoring and Adaptation in heterogeneous environments (MonAdapter). The architectural design of Mon-Adapter is depicted in Figure 2.

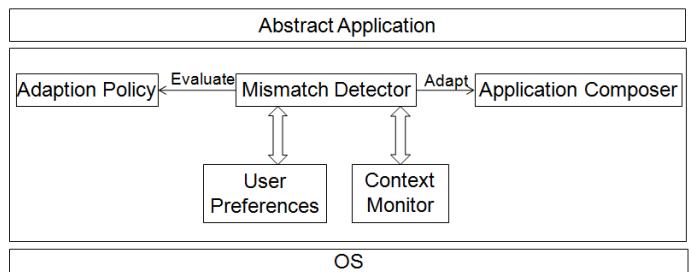


Fig. 2. Middleware for Monitoring and Adaptation

The middleware consists of a taskResolver component that maps the user task to the concrete level to identify components provided by the available devices. To that end, it relies on the device selection and component selection services to resolve the user task by mapping it to the concrete components based on the user preferences and some non-functional aspects [5].

A Mismatch Detector component is used to analyze the abstract description of the application compared to the capabilities of the selected devices, the user preferences and the extra-functional requirements of the components. For this goal, it relies on some monitor to capture the changes of the user preferences and the execution environment (network status, arrival and departure of devices or components, etc).

In case of the failure of the mapping or the application's execution due to the changes in the context or user preferences, the Mismatch Detector evaluates the application composition according to the adaptation policies provided by the Adaptation Policy component. If a policy for adaptation exist for that mismatch, the Application Composer is informed to adapt the application accordingly.

III. STRUCTURAL ADAPTATION APPROACH

A. Principle of Our Approach

To overcome a captured mismatch between an abstract application and the concrete level, there is a need to adapt the abstract application to ensure its mapping and its execution.

Therefore, in our previous work [4], we have proposed a dynamic structural adaptation approach for abstract applications.

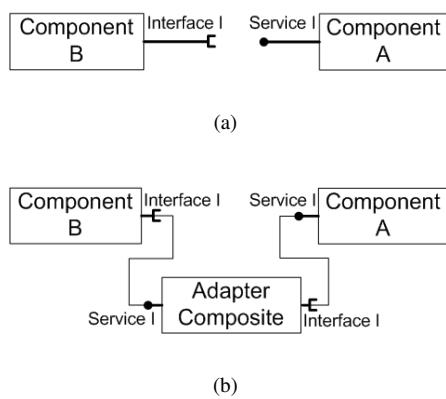


Fig. 3. Transforming abstract application using Adapter composite

Our approach consists of transforming an abstract application to another one that allows its mapping and execution. The transformation is ensured by injecting some extra-functional adapters into the abstract application without modifying its functional behavior.

For example, as shown in Figure 3, an adapter composite is injected to adapt the communication between components A and B. The adapter composite requires the service I of the component A and exposes a service implementing the interface I. This provided service will be used by the component B, since it corresponds to its required service. Thus, the abstract application is transformed by adding an extra-functional adapter to achieve its mapping or its execution.

B. Adaptation Patterns

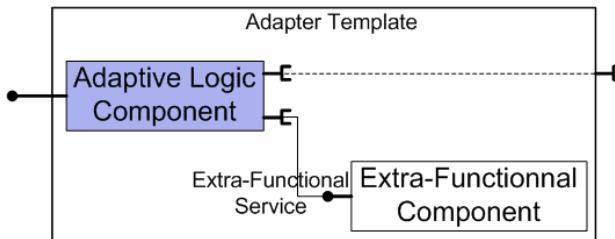


Fig. 4. Generic adapter template

As the basis for our approach, we have proposed to use adaptation patterns as adapter composites, which provide solutions for the detected mismatches between an abstract application and a concrete level and can be used in different contexts [4].

An adaptation pattern is defined following an adapter template, which consists of an adaptive logic and extra-functional components as shown in Figure 4. The extra-functional component provides transformation services allowing, e.g., encryption, compression, etc. However, the adaptive logic component encapsulates the adaptation logic and acts as an intermediate between the abstract and the extra-functional components. This component has a generated implementation as it depends to the interfaces of communicated components.

Using this approach allows us to separate the extra-functional logic from the business one, and hence, to add or remove adaptation patterns dynamically from the abstract application whenever there is a need.

In the following, we present a set of adaptation patterns that are defined following the adapter template. For each pattern, we present its description, the context in which, it will be used, and where it will be used to overcome that mismatch.

The list of the adaptation patterns is not exhaustive. However, it is possible to define such other patterns following our adapter template.

1) Encryption and Decryption patterns:

a) Description: we propose an encryption pattern that intercepts the interaction between components to encrypt messages transiting over a network in order to prevent the disclosure of information to unauthorized components. To use the original message, there is a need to restore it from the encrypted one. For this purpose, we have composed the encryption adapter with a decryption one that decrypts the received messages before using it by the target component.

The encryption and decryption adapters are defined following our adapter template. Each adapter consists of an adaptive logic component and an extra-functional one as shown in the Figure 5. The extra-functional component exposes a key property and provides a service ensuring encryption or decryption of a message following a symmetric key algorithm.

In case of a symmetric encryption and decryption, the transmitter and the receiver sides use the same key to exchange messages. However, if the extra-functional components implement an asymmetric algorithm, they use two different keys: public and private key.

b) Adaptation context: The encryption and decryption patterns will be injected at init time to ensure the security of the transferred messages, which may be sensitive to disclose as with credit card numbers and passwords.

The security requirement can be expressed explicitly through the non-functional requirements of services. If the concrete components do not ensure this requirement as specified in the abstract level, there is a need to adapt the application in order to achieve its mapping.

For this purpose, the encryption and decryption patterns will be used to overcome the non-satisfaction of the non-functional requirements of services.

c) Where to use: The encryption and decryption patterns are used in distributed manner; the transmitter device will contain the encryption adapter to send encrypted messages, while the decryption pattern is used by the receiver side to restore the messages. For example, in Figure 5, an encryption pattern is used by a device B in order to encrypt the messages sent from a component B over the network. However, a decryption pattern was handled in a device A to restore the original message before using it by a component A.

2) Authentication and Integrity patterns:

a) Description: The authentication pattern is used to sign the transferred message between two components in order to ensure that a message has not been tampered with and that

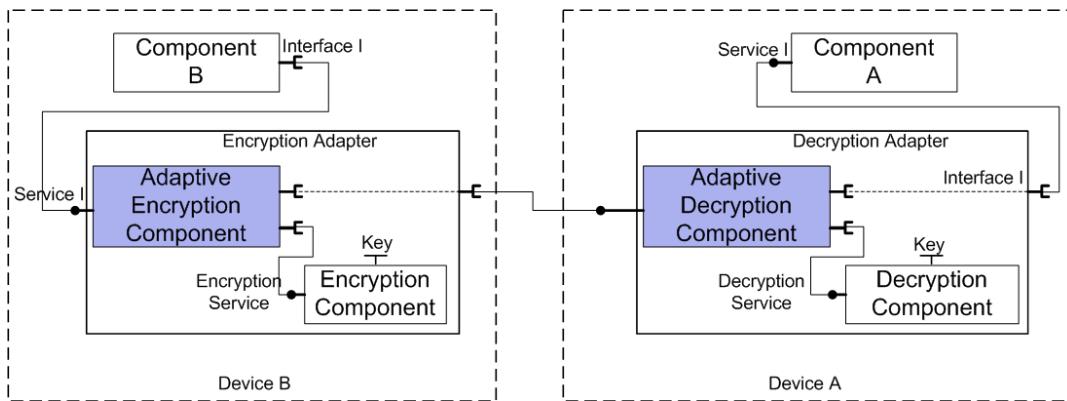


Fig. 5. Encryption and Decryption adaptation patterns

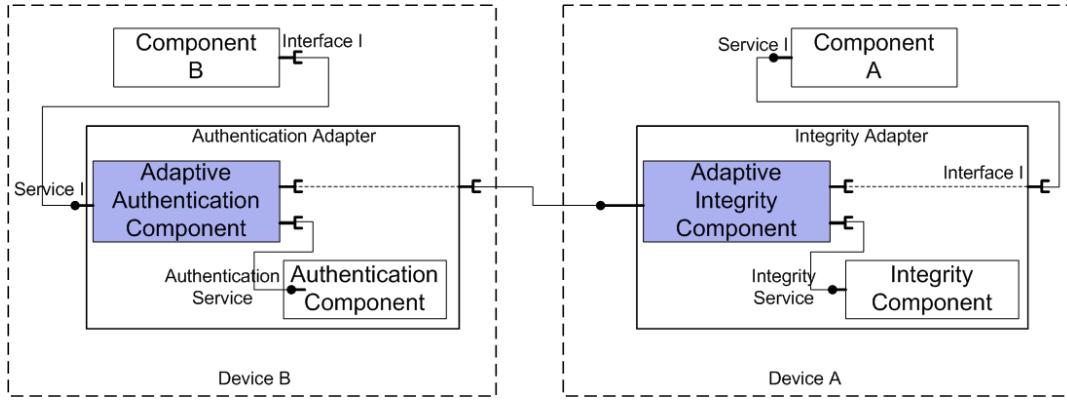


Fig. 6. Authentication and Integrity adaptation patterns

it originated from a certain component. The extra-functional component of the pattern generates a signature digest to add it at the end of the message before sending this later over the network.

In the receiver side, there is a need to validate the authentication of the message's signature to authorize component to invoke the requested service. Therefore, we have composed it with an integrity pattern that will prove the validity of a the transmitted message before forwarding it to the intended component.

The verification is done by comparing the received digest with a calculated one. If the message digest of the message matches the message digest from the signature, then integrity is also assured. Otherwise, the message is tampered with during its transfer. Thus, the extra-functional component of the integrity component, in Figure 6, returns a boolean result implying the validity of signed messages.

b) Adaptation context: An abstract component may specify at init time through its non-functional requirements the need to validate the received messages. If the concrete component does not ensure the authentication and the integrity of messages, this implies a mismatch between the abstract description and the concrete level.

c) Where to use: The authentication pattern is used by the transmitter side to send signed messages. In the receiver

side, the integrity pattern will be used to check if the message is kept intact during its transfer over the network. Figure 6 shows an authentication pattern that is used by device B to send signed message from a component B to a component A. However, an integrity pattern is used by the device A, to validate the received message from the component B.

3) Splitting and Merging patterns:

a) Description: The splitting pattern is used to split a message into chunks. The pattern consists of an adaptive logic and an extra-functional component that returns a list of chunks to send over the network as shown in Figure 7.

To respond to the component's request, there is a need to merge the chunks beforehand. Therefore, we propose to compose the splitter pattern with a merger one to form the message from the received chunks. Hence, the extra-functional component of the merger pattern will construct messages from the received chunks, which are forwarded by the adaptive logic component to the intended component.

b) Adaptation context: The splitting and merging patterns are used to overcome a problem related to a lower network signal in order to have a decreased message delay. This may have an impact certainly for the transfer of bigger files or messages. In this case, the message is split into chunks for a quick transfer over a network and then merged to construct the original message.

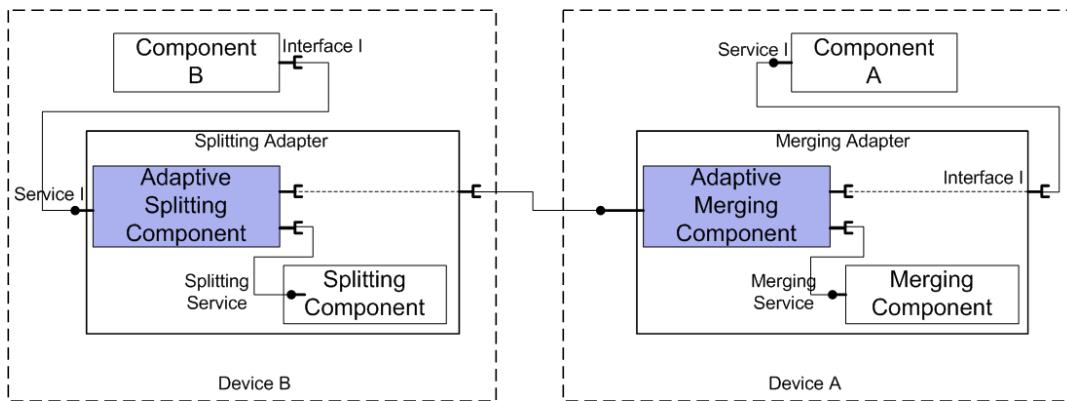


Fig. 7. Splitting and Merging adaptation patterns

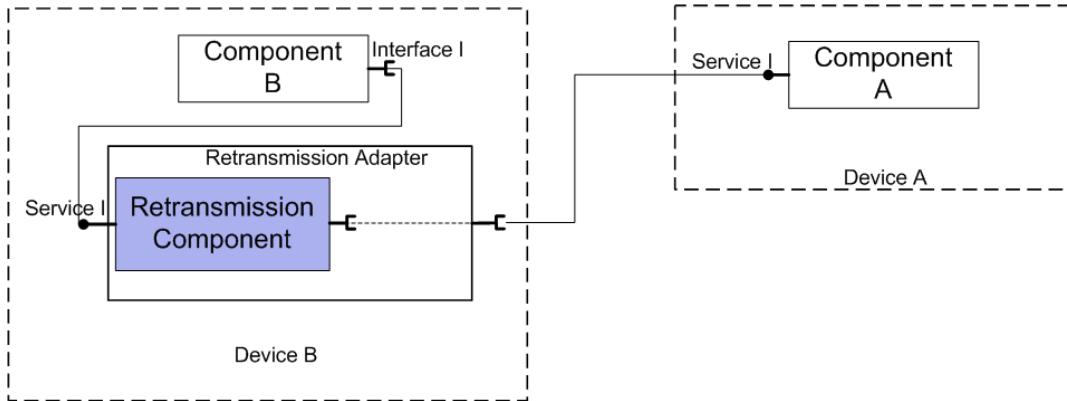


Fig. 8. Retransmission adaptation pattern

c) *Where to use:* The splitting pattern will be used by the transmitter device, while the merging pattern will be used by the receiver device to fulfill the interaction between devices. Thus, the component B in Figure 7 is able to send a message or a file into chunks to the component A for a quick transfer over a lower network signal.

4) Retransmission pattern:

a) *Description:* This pattern provides the functionality of retransmitting a failed call to a remote component. Its adaptive logic component, as shown in the Figure 8, attempts to retransmit message whenever a component does not receive a response to its calls. The component may use some error-detection codes or acknowledgments to achieve reliable message retransmission.

b) *Adaptation context:* The transmission in pervasive environment may be subject to failures because of e.g. network down. This can be captured by tracking the network work for a period of time. If the statistics shows that the system is not reliable, thus, the components' messages could get lost along the path. When it is not possible to deliver messages to remote components, the system should attempt to respond to the component request at the earliest possible opportunity by trying to retransmit the messages. For this reason, we propose a retransmission pattern that attempts to retransmit the messages to render the application reliable at init time.

The retransmission pattern is used also during the execution of the application, if the network is quick cut-off, to overcome the loss of messages. Thus, once the network is repaired, the retransmission pattern is established to retry the sending of calls. To detect this adaptation context, the middleware should monitor the status of the supported network, i.e., if it is activated or not.

c) *Where to use:* To ensure a reliable communication between devices, we propose to handle at init time a retransmission pattern in the sender side. For example, Figure 8 shows a retransmission pattern that is used by a device B to resend the failed call to a device A.

5) Compression and Decompression Patterns:

a) *Description:* The compression pattern is introduced between two components communicating with each other over a network in order to send compressed messages. However, in the receiver device, there is a need to decompress the message in order to be used by the target component. Therefore, we propose to compose the compression pattern with a decompression one to decompress the data before using it by the target component.

Each adapter consists of an adaptive logic component that relies on a non-functional component to compress or decompress message, as shown in Figure 9.

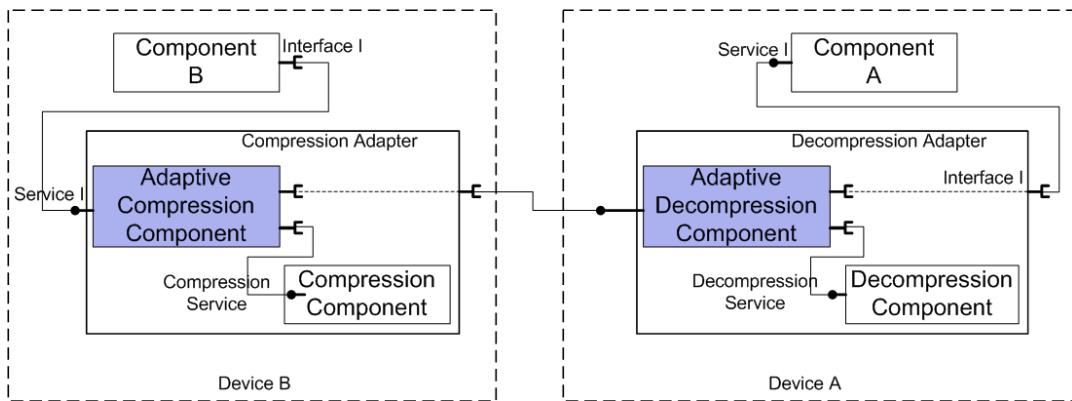


Fig. 9. Compression and Decompression adaptation patterns

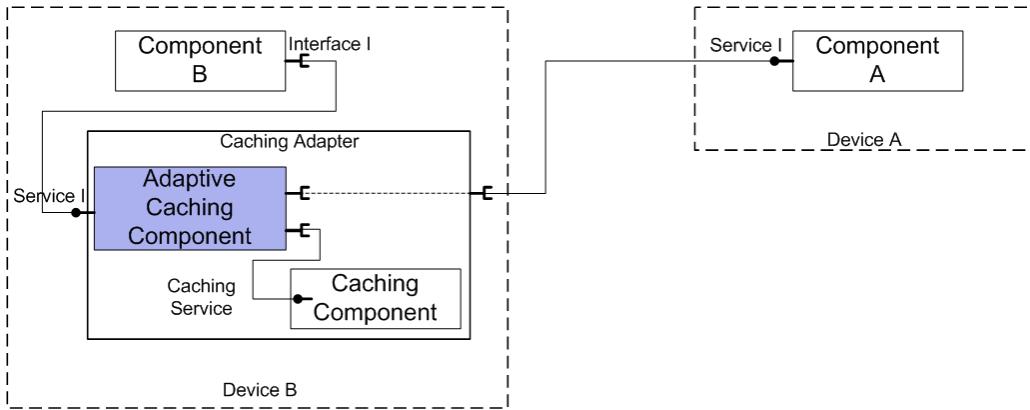


Fig. 10. Caching adaptation pattern

b) Adaptation Context: The compression and decompression patterns are introduced in response to a trigger generated by fluctuation in network QoS. For example, two components exchanging data may be adapted by using the compressor and decompressor adapters if the network latency or the throughput falls below a certain threshold. By using this adapter composite between the components, all the data will be compressed before transmission over network, allowing efficient transfer of data.

c) Where to use: We propose to use the compression pattern by the sender device in order to send compressed messages over network. However, the decompression pattern should be handled in the receiver side to decompress messages before using it as shown in Figure 9.

6) Caching Pattern:

a) Description: The caching pattern enables the application to cache messages in rapid memory. Figure 10 shows the main features of the caching pattern. It consists of an adaptive logic component that will first check the cache to see for example if the response of the component request can be found there. Failing to find the response in the cache, the adaptive caching component will forward the call to the target component. Once it receives the response, it will forward it to the caller component after storing it in the cache.

Thus, the caching service provides mainly methods for

retrieving, updating and setting message in the cache. We assume also that the cache is already created else, the caching service should be able also to create a cache in a device.

b) Adaptation context: The caching pattern can be used during the execution of the application to avoid the congestion of a network by storing the responses to the services' requests. Therefore, the system should monitor the latency of the used network to identify if there is not a jitter. Otherwise, a caching pattern will be injected during the execution application to avoid the congestion for a further uses.

Moreover, some component may express through their non-functional requirements the need to have a decreased response time, for example the response time of service I is less than 50 ms. If the concrete component does not consider this requirement, there is a need to inject a caching pattern at init time in order to decrease the response time during the execution of an application.

c) Where to use: The caching pattern will be used either by the sender or the receiver side where the message will be stored. For example, in Figure 10, the pattern is used to decrease the response time to the requests of the component B by caching the call to service I in a cache of the device B.

7) Proxy Pattern:

a) Description: The proxy pattern allows components to access to services offered by others components. Figure 11

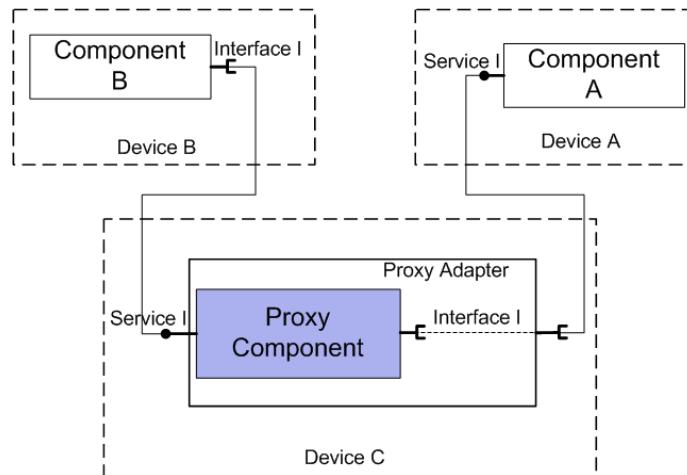


Fig. 11. Proxy adaptation pattern

shows a description of the proxy pattern following the adapter template. As it can be seen, the proxy pattern represents a specific case of the adapter template as it contains only an adaptive logic component that forwards the call of the service I to the component A.

b) *Adaptation Context:* The proxy pattern is useful to overcome the network factor related to the heterogeneity of network interfaces. For example, if two devices were selected to map an abstract application and they support two different connection interfaces, e.g., Bluetooth and Wifi, thus, the mapping will fail. Therefore, we propose to introduce the proxy pattern to act as an intermediate between the communicating components.

c) *Where to use:* To intermediate the communication between devices, we propose to generate the proxy in a third device. Thus, the components A and B, as shown in Figure 11, can communicate together via the proxy generated in a device C.

IV. EXAMPLE SCENARIO USING ADAPTATION PATTERNS

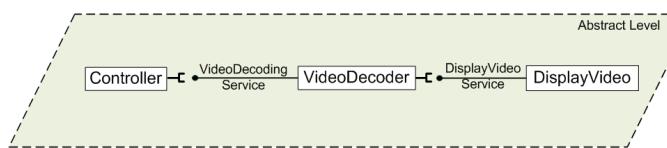


Fig. 12. Video Player application

Referring back to the video player application described in the introductory section, Figure 12 shows an abstract description of the Video player application that consists of three components: a *VideoDecoder* component, a *DisplayVideo* component and a *Controller* Component. The *Controller* component sends a command to the *VideoDecoder* component to decode a stored video. The *VideoDecoder* component decodes a video into appropriate format. Once the video is decoded, it is passed to the *DisplayVideo* component to play it. This is

done using the service provided by the *DisplayVideo* component through an appropriate programming interface. The description of an application can be done with the help of an Architecture Description Language (ADL). For this purpose, we have used Service Component Architecture (SCA) [18] to describe abstract applications.

SCA provides a programming model for building applications and systems based on a Service Oriented Architecture. The main idea behind SCA is to be able to build distributed applications, which are independent of particular technology, protocol, and implementation. SCA applications are deployed as composites. An SCA composite describes an assembly of heterogeneous components, which offer functionality as services and require functionality from other components in the form of references. Along with services and references, a component can also define one or more properties.

Figure 13 shows an SCA description of the *VideoPlayer* application shown previously in Figure 12. It provide *DisplayVideoService* and consists of the controller, *videoDecoder* and *DisplayVideo* abstract components.

Using SCA, it is also possible to specify the services' requirements abstractly and the components' implementations provide the corresponding concrete policies [17]. Abstract resource requirements can be specified by using @requires attribute, while the @policySets attribute is used to specify the concrete resources. The policies are applied to implementation and contain the requirements that should be fulfilled before selecting the components to which the policy sets are attached.

For example, the controller component requires that its messages sent to the *DecodingVideoService*, should be authenticated. Therefore, its reference is marked with an intent "authentication" (line 4 in Figure 13). However, the *DecodingVideoService* is marked with the "integrity" intent to check the validity of the messages received from the controller component (line 7 in Figure 13). Figure 14 shows a description of the authentication abstract intent that is applied to the component binding.

```

1<composite name="VideoPlayer">
2  <service name="DisplayVideoService" promote="DisplayVideoComponent/DisplayVideoService" />
3  <component name="ControllerComponent">
4    <reference name=" DecodingVideoService" target="VideoDecoderComponent" requires="authentication"/>
5  </component>
6  <component name="VideoDecoderComponent">
7    <service name="DecodingVideoService" requires="integrity">
8      <interface.java interface="eu.tsp.iaria-example.VideoDecoderInterface" >
9    </service>
10   <reference name="DisplayVideoService" target="DisplayVideoComponent " />
11 </component>
12 <component name="DisplayVideoComponent">
13   <service name="DisplayVideoService">
14     <interface.java interface="eu.tsp.iaria-example.DisplayVideoInterface" />
15   </service>
16 </component>
17 </composite>

```

Fig. 13. SCA description of the Video Player Application

```

<intent name="authentication" constrains="sca:binding">
<description>
Communication through this binding must be authenticated.
</description>
</intent>

```

Fig. 14. SCA policy intent of an authentication requirement

The resolution of the abstract description of the video player application into respective concrete components is required for the realization of the task. We assume that the execution environment consists of three devices: a Smartphone (SP), a flat-screen (FS) and a laptop device (LP).

Following the matching algorithm [16], the application composer of the middleware has identified a *Controller* and the *VideoDecoder* components in SP and a *DisplayVideo* component in FS. Thus, the LP device is eliminated.

However, the concrete components of the *videoDecoder* and the controller services do not support the policies related to the authentication and integrity intents as specified in Figure 13. Thus, there is a mismatch between the given abstract description of the video player application and the concrete level.

To resolve this mismatch, we propose to inject the authentication and integrity patterns into the abstract application as shown in Figure 15. Thus, the controller component is able to send authenticated commands to the *VideoDecoder* component. These commands will be validated at first by the integrity pattern before forwarding it to the *videoDecoder* component. As a result, the application is transformed, as shown in figure 15, to contain the authentication, and the integrity patterns in addition to its own components.

Figure 16 represents the SCA description of the integrity pattern. It consists of an adaptive logic component representing the integrity intent. To this end, we have extended the SCA description by the @type attribute (line 6) to specify what is

the intent represented by the adaptive logic component if it is either a proxy or a splitting or a compression patterns. Moreover, the implementation of the adaptive logic component should be generated dynamically since this component depends to the *decodingVideoService* of the *videoDecoder* component (line 9). For this purpose, we have extended SCA by a new attribute @generated that specifies if the implementation is generated or not (line 6). Furthermore, the adaptive logic component relies on the integrity extra-functional component to check the validity of the received message before forwarding it to the *VideoDecoder* component.

In another case, we assume that during the execution of the application, the bandwidth of the supported network becomes weak. This may have an impact on the quality of video that requires a high QoS. Towards this change of context, we propose to adapt the abstract video player application by splitting the frame into chunks and then compress them for a quick transfer.

For this purpose, we have composed together the splitting, compression, decompression and merging patterns, as shown in Figure 15 for a quick transfer of messages with a lower bandwidth. The splitter adapter will split a frame sent from the *VideoDecoder* component to the *DisplayVideo* one into chunks. These latter will be compressed before their transfer over the network. Once the chunks are received by a device, there is a need at first to decompress them and then to merge them before forwarding it to the *DisplayVideo* component. Hence, the abstract application is adapted by injecting a composite of adapters to overcome a mismatch triggered by a low bandwidth.

V. IMPLEMENTATION

In order to validate our approach, we have implemented a prototype in Java. To that end, we have used SCA [18] to describe applications in abstract way and then map them to the concrete components.

The open source software JAVA programming ASSISTant (Javassist) library [11] is used to generate the byte codes of the

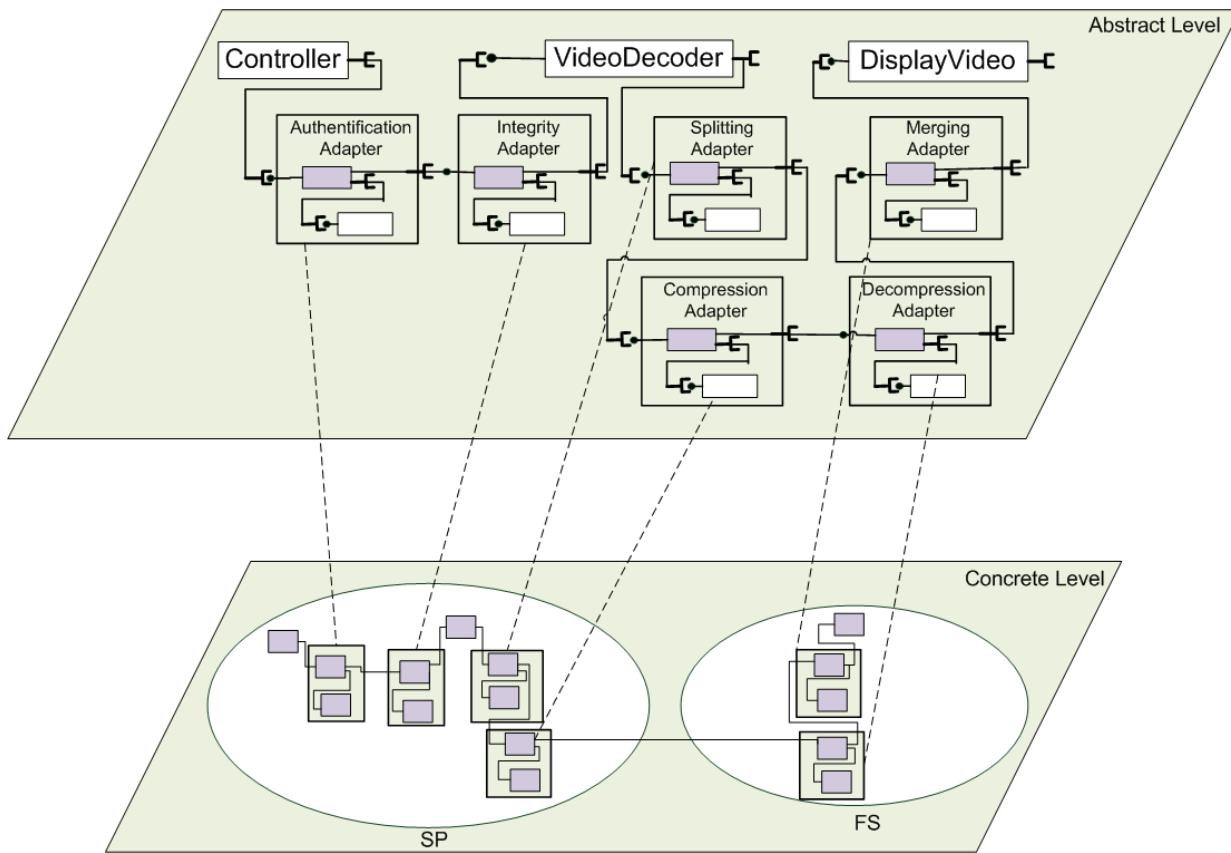


Fig. 15. Adaptation of the Video Player application

adaptive logic component of the adaptation patterns. Indeed, it enables Java programs to define a new class at runtime and to modify a class file when the Java Virtual Machine (JVM) loads it.

Moreover, the java API `java.lang.reflect` is used to obtain reflective information about classes and objects. This information is used by the Javassist library to allow the adaptive logic component to implement the required service.

VI. RELATED WORK

A lot of work has been devoted to structural adaptation of component-based applications due to mismatches captured at the concrete level. In the following, we detail some of the existing approaches as well as their limitations.

Spalazzese and Inverardi [19] considers a mediator concept to cope with the heterogeneity of the application-layer protocols. The approach first abstracts the behavioral description of the mismatching protocols highlighting some structural characteristics. This is done by using ontology. Then, it checks the possibility for the two protocols to communicate. If the two protocols are not complementary, the framework should find out the suitable mediating connector using some basic mediators connectors patterns. However, these mediators connectors are limited to the protocol level. Moreover, the mediators are specified only by the framework and their concrete realizations remain a challenge for the authors.

In the same context, Fuentes et al. propose to use aspectual connectors that provide support to describe and to weave aspects to components [8]. However, these connectors are described at design time, which limits the possibility to extend applications with new aspects. Moreover, the specification of connector template relies on the used aspects as well as the functional behavior of application.

In [13], Li and al. tackle the behavioral mismatches and propose to use the mediation patterns. They categorize the mismatching levels and focus their work on signature and protocol mediations using six basic patterns. The identification of pattern is done following some rules predefined by the designer.

The major drawback of this approach is that is limited to the behavioral mismatches, thus, they do not consider the mismatches related to hardware, network characteristics of devices. Moreover, the generation of the mediator pattern is done by pseudo codes predefined by the designer. However, in our work, the adaptive logic component of the adaptation pattern is generated dynamically by specifying only its required, provided interfaces and the extra-functional service.

Other related work in this area [7][12] have also investigated matching of Web service interfaces by providing a classification of common mismatches between service interfaces and business protocols, and introducing mismatch patterns. These patterns are used to formalize the recurring problems

```

1 <composite name="IntegrityAdapter">
2   <service name = "DecodingVideoService" promote= "AdaptiveLogicComponent/DecodingVideoService" >
3     <component name="AdaptiveLogicComponent" >
4       <service name="DecodingVideoService">
5         <interface.java interface="eu.tsp.iaria-example.VideoDecoderInterface" />
6         <implementation.java type ="Integrity" generated="True" />
7       </service>
8       <reference name="IntegrityComponent"/>
9       <reference name=" DecodingVideoService" target="VideoDecoderComponent" />
10    </component>
11    <component name="IntegrityComponent">
12      <service name="IntegrityService">
13        <interface.java interface="eu.tsp.iaria-example.IntegrityInterface" />
14      </service>
15    </component>
16 </composite>

```

Fig. 16. SCA description of the integrity pattern

related to the interactions between services. The mismatch patterns include a template of adaptation logic that resolves the detected mismatch. Developers can instantiate the proposed template to develop adapters. For this purpose, they have to specify the different transformation functions.

We can identify two important limitations compared to our approach. First, the mismatch patterns are limited to the interfaces and protocols mismatches. Second, the specification of adapters supplies some pseudo code predefined by the designer. However, in our approach, we are able to specify dynamically the different components of a used pattern; by generating the implementation of its adaptive logic component and mapping the extra-functional one following our matching algorithm.

In [10], Cao et al. propose an approach to component adaptation dealing with non-functional mismatches. Their adaptation framework includes extra-functional adapters which mediate the mismatching behaviors between the client and server. Therefore, they propose to use adapters presented that provide extra-functional interfaces customized by the user.

Compared to our approach, the specification of adapters is done with the help of the user, whereas in our work, the adapters are specified using our template. Moreover, the specified adapters depend to the adapted application. Hence, it can be used on other context. However, our patterns may be used by any applications as their description is independent of the functional behavior of the application.

VII. CONCLUSION AND FUTURE WORK

In this paper, we have identified some situational contexts at init time and during the execution of the application, according to which the application can be adapted. These contexts represent mismatches between an abstract application and the concrete level and they may arise at inter-components, intra-device or inter-devices levels.

Towards these mismatches, we have proposed a set of adaptation patterns that are injected into an abstract application to ensure its mapping or its execution. These adapters provide an extra-functional behavior with respect to the functional

behavior of the application. The list of the adaptation patterns is not exhaustive. However, it is possible to define such other patterns following our adapter template.

We are looking forward to identify rules for the use of adaptation pattern describing where and when the adapter will be used.

REFERENCES

- [1] Gregory D. Abowd, Anind K. Dey, Peter J. Brown, Nigel Davies, Mark Smith, and Pete Steggles. Towards a better understanding of context and context-awareness. In *the 1st international symposium on Handheld and Ubiquitous Computing*, HUC' 99, pages 304–307, Karlsruhe, Germany, 1999.
- [2] Christian Becker, Marcus Handte, Gregor Schiele, and Kurt Rothermel. Pcom - a component system for pervasive computing. In *Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications*, PerCom'04, page 67, Orlando, FL, USA, 2004.
- [3] Steffen Becker, Antonio Brogi, Sven Overhage, Er Romanovsky, and Massimo Tivoli. Towards an engineering approach to component adaptation. In *Springer-Verlag, LNCS*, page 2006. Springer, 2006.
- [4] Imen Ben Lahmar, Djamel Belaïd, and Hamid Mukhtar. Adapting abstract component applications using adaptation patterns. In *Proceedings of the Second International Conference on Adaptive and Self-adaptive Systems and Applications*, ADAPTIVE, pages 170–175, Lisbon Portugal, 2010.
- [5] Imen Ben Lahmar, Djamel Belaïd, Hamid Mukhtar, and Sami Chaudhary. Automatic task resolution and adaptation in pervasive environments. In *Proceedings of the Second International Conference on Adaptive and Intelligent Systems*, ICAIS, pages 131–144, Klagenfurt, Austria, 2011.
- [6] Sonia Ben Mokhtar, Nikoleta Georgantas, and Valérie Issarny. Cocoa: Conversation-based service composition in pervasive computing environments with qos support. *Journal Of System and Software*, vol 80, no 12:1941–1955, 2007.
- [7] Boualem Benatallah, Fabio Casati, Daniela Grigori, H. R. Motahari Nezhad, and Farouk Toumani. Developing adapters for web services integration. In *Proceedings of the International Conference on Advanced Information Systems Engineering*, CAiSE, pages 415–429, Porto, Portugal, 2005.
- [8] Lidia Fuentes, Nadia Gámez, Mónica Pinto, and Juan A. Valenzuela. Using connectors to model crosscutting influences in software architectures. In *The First European Conference on Software Architecture*, ECSA'07, pages 292–295, Madrid, Spain, 2007.
- [9] Matthias Galster and Eva Bucherer. A taxonomy for identifying and specifying non-functional requirements in service-oriented development. In *IEEE Congress on Services*, SERVICES '08, pages 345–352, Honolulu, Hawaii, USA, 2008.

- [10] Jingang Xie Guorong Cao, Qingping Tan. A new approach to component adaptation dealing with extra-functional mismatches. In *International Conference on Information Engineering and Computer Science*, Wuhan, China, 2009.
- [11] JAVA programming Assistant. <http://www.csg.is.titech.ac.jp/chiba/javassist/>.
- [12] Woralak Kongdenpha, Hamid Reza Motahari-Nezhad, Boualem Benatallah, Fabio Casati, and Regis Saint-Paul. Mismatch patterns and adaptation aspects: A foundation for rapid development of web service adapters. *IEEE Transactions on Services Computing*, pages 94–107, 2009.
- [13] Xitong Li, Yushun Fan, Stuart Madnick, and Quan Z. Sheng. A pattern-based approach to protocol mediation for web services composition. *Information and Software Technology (IST)*, 52:304–323, March 2010.
- [14] Philip K. McKinley, Seyed Masoud Sadjadi, Eric P. Kasten, and Betty H.C. Cheng. Composing adaptive software. *Journal of IEEE Computer*, 37:56–64, 2004.
- [15] Hamid Mukhtar, Djamel Belaïd, and Guy Bernard. A graph-based approach for ad hoc task composition considering user preferences and device capabilities. In *Workshop on Service Discovery and Composition in Ubiquitous and Pervasive Environments*, New Orleans, LA, USA, dec 2007.
- [16] Hamid Mukhtar, Djamel Belaïd, and Guy Bernard. User preferences-based automatic device selection for multimedia user tasks in pervasive environments. In *the 5th International Conference on Networking and Services*, ICNS' 09, pages 43–48, Valencia, Spain, 2009.
- [17] Open SOA Collaboration. Sca policy framework v1.00 specifications. <http://www.osoa.org/>, 2007.
- [18] Open SOA Collaboration. Service component architecture (sca): Sca assembly model v1.00 specifications. <http://www.osoa.org/>, 2007.
- [19] Romena Spalazzese and Paola Inverardi. Mediating connector patterns for components interoperability. In *The fourth European Conference on Software Architecture*, ECSA'10, pages 335–343, Copenhagen, Denmark, 2010.

On the Quality of Relational Database Schemas in Open-source Software

Fabien Coelho, Alexandre Aillois, Samuel Pilot, and Shamil Valeev

CRI, Mathématiques et Systèmes, MINES ParisTech,

35, rue Saint Honoré, 77305 Fontainebleau cedex, France.

fabien.coelho@mines-paristech.fr, firstname.lastname@mines-paris.org

Abstract—The relational schemas of 512 open-source projects storing their data in MySQL or PostgreSQL databases are investigated by querying the standard *information schema*, looking for overall design issues. The set of SQL queries used in our research is released as the *Salix* free software. As it is fully relational and relies on standards, it may be installed in any compliant database to help improve schemas. Our research shows that the overall quality of the surveyed schemas is poor: a majority of projects have at least one table without any primary key or unique constraint to identify a tuple; data security features such as referential integrity or transactional back-ends are hardly used; projects that advertise supporting both databases often have missing tables or attributes. PostgreSQL projects appear to be of higher quality than MySQL projects, and have been updated more recently, suggesting a more active maintenance. This is even better for projects with PostgreSQL-only support. However, the quality difference between both databases management systems is mostly due to MySQL-specific issues. An overall predictor of bad database quality is that a project chooses MySQL or PHP, while good design is found with PostgreSQL and Java. The few declared constraints allow to detect latent bugs, that are worth fixing: more declarations would certainly help unveil more bugs. Our survey also suggests that some features of MySQL and PostgreSQL are particularly error-prone. This first survey on the quality of relational schemas in open-source software provides a unique insight in the data engineering practice of these projects.

Keywords-open-source software; database quality survey; automatic schema analysis; relational model; SQL.

I. INTRODUCTION

This paper is an extended version of *A Field Analysis of Relational Database Schemas in Open-source Software* [1] presented at DBKDA 2011. Compared to this initial version, 512 schemas are surveyed instead of 407, which enhances the accuracy of the statistical validation of our analyses; the maintenance status of the surveyed projects was collected again as of January 2012; comments have been updated and added to reflect the new data; more detailed tables are provided about the results; the bibliography is much more thorough, with over 50 new references; an appendix describes the advices available with our schema analyzer; the paper page count, excluding the appendix, is increased from 7 to 10 pages.

In the beginning of the computer age, software was freely available, and money was derived from hardware only [2]. Then in the 70s it was *unbundled* and sold separately in closed proprietary form. Stallman initiated the free software movement, in 1983 with the *GNU Project* [3], and later the

Free Software Foundation [4], which is now quite large [5][6] and expanding [7] (Predicts 2010) to implement his principle of sharing software. Such free software is distributed under a variety of licenses [8], which discuss copyright and liability. The common ground is that it must be available as source code to allow its study, change and improvement as opposed to compiled or obfuscated, hence the expression *open source* [9][10][11]. This induces many technical, economical, legal, and philosophical issues. Open-source software (OSS) is a subject of academic studies [12] in psychology, sociology, economics, or software engineering, including quantitative surveys. Developers' motivation [13][14][15][16][17], but also organization [18][19][20][21][22][23][24][25][26] and profiles [27][28][29] are investigated, as well as user communities [30]; Existing economic frameworks [31] are used to analyze the phenomenon, as well as the influence of public policies [32]. Research focusing on software engineering issues can also be found. The development of the Apache web server popular [33] is compared to non-OSS projects [34] and its user assistance is analyzed [35]. Quantitative studies exist about code quality in OSS [36][37][38][39][40] and its dual, static analysis to uncover bugs [41][42]. Database surveys are available about market shares [43], or server exposure security issues [44]. This study is the first survey on the quality of relational database schemas in OSS. It provides a unique insight in the data engineering practice of these projects.

Codd's relational model [45] is an extension of the set theory to relations (tables) with attributes (columns) in which tuple elements are stored (rows). Elements are identified by keys, which can be used by tuples to reference one another between relations. The relational model is sound, as all questions (in the model) have corresponding practical answers and vice versa: the tuple relational calculus describes questions, and the mathematically equivalent relational algebra provides their answers. It is efficiently implemented by many commercial and open-source software such as Oracle, DB2 or SQLite. The *Structured Query Language* (SQL [46]) is available with most relational database systems, although the detailed syntax often differs in subtle and incompatible ways. The standardization effort also includes the *information schema* [47], which provides metadata about the schemas of databases through relations.

The underlying assumption of our study is that applications store precious transactional user data, thus should be kept consistent, non redundant, and easy to understand. We think that

database features such as key declarations, referential integrity and transaction support help achieve these goals. In order to evaluate the use of database features in open-source software, and to detect possible design or implementation errors, we have developed a tool to analyze automatically the database structure of an application by querying its *information schema* and generating a report, and we have applied it to 512 open-source projects. The notion of the quality of a database schema design is quite elusive, as shown in Burkett's overview [48], with a lot of focus on qualitative assessments. Key criteria such as understandability, simplicity, expressiveness, maintainability or evolvability are hard to transform into basic objective metrics. A review process has been proposed to evaluate the quality of relational schemas [49], at the price of mostly manual investigations by field experts. Some quality focus on the conceptual schema and compare alternative models [50][51] by recognizing patterns. Following MacCabe's metric to measure automatically program complexities [52][53][54], several metrics address data models [55][56] or database schemata either in the relational [57][58] or object relational [59] models, including experimental validations [60]. These metrics rely on information not necessarily available from the database concrete schemas. Moreover, such approach help compare two schemas that model the same application domain, but are less useful when used about unrelated schemas. We have rather followed the dual and pragmatic approach [61], which is not to try to do an absolute and definite measure of the schema, but rather to uncover issues based on static analyses. Thus, the measure is relative to the analyses performed and results change when more are added. Static analysis on user application codes (not simply the schema) could also be used to help uncover hidden constraints in a schema (for instance, a join between two tables suggests a possible foreign key) and to use them to improve data quality [62], but this is beyond our simple approach.

The remainder of this paper is organized as follows: Section II presents the methodology used in this study. We describe our tool, our rating strategy and the statistical validation used on the assertions derived from our analyses; Section III lists the projects by category and technology, and discusses similarities and differences depending on whether they run on MySQL or PostgreSQL; Section IV describes the results of our survey, with quite a poor overall quality of projects, as very few database schemas do not raise error-rated advices; Section V gives our conclusive thoughts.

II. METHODOLOGY

Our *Salix* automatic analyzer [63], is based on the *information schema* provided by standard databases. It is open-source, and its schema itself is included in this survey. In this Section, we discuss the queries, then describe the available advices, before presenting the statistical validation used.

A. Information schema queries

Our analyses are performed automatically by SQL queries on the databases metadata using the standard *information schema*. This relational schema stores information about the

databases structure, including catalogs, schemas, tables, attributes, types, constraints, roles, permissions, etc. The set of SQL queries used for this study are released as the *Salix* free software. It is based on *pg-advisor* [64], a PostgreSQL-specific proof of concept prototype developed in 2004. Some checks are inspired by Currier [65], Baron [66] and Berkus [67] or similar to Boehm [68]. Note that the aim is quite different from tools which focus on advising database administrators, for instance about index creation [69]. *Salix* creates specific tables for each advice by querying the *information schema*, and then aggregates the results in summary tables in a dedicated schema. It is fully relational in its conception [70]; there is no programming other than SQL queries, but a small shell driver which *creates* the advices, *shows* or *reports* them in some detail to the interested user, and finally *drops* them out of the database. Because of performance issues when querying heavily metadata relations, the tool relies on tables which are materialized views, although using views directly would have been a preferred option if possible. The development of *Salix* uncovered multiple issues with both implementations of the *information schema*.

B. Advice classification and project grading

The 47 issues reported by our SQL queries from the standard *information schema* are named *advices*, as the user is free to ignore them. Although the performed checks are basic and syntactic, we think that they reflect the quality of the schemas. For instance, style advices help with understandability, and consistency advices help with maintainability. A detailed list of advices currently implemented in our tool is available [71]. Each advice has a category (19 design, 13 style, 6 consistency, 4 version, 5 system), a severity (7 errors, 21 warnings, 14 notices, 5 informations), and a level (1 raised per database, 10 per schema, 27 per relation, 7 per attribute, 2 per role). The severity classification is arbitrary and must be evaluated critically by the recipient: most of them should be dealt with, but in some cases they may be justifiable. For instance, having a mix of MySQL back-end engines is considered inconsistent and tagged as an error, although it may be necessary to do so because some features (*e.g.* full-text indexes) are only available with some back-ends. Moreover, detected errors do not imply that the application is not fully functional from a user perspective.

The 19 **design** advices focus on detecting design errors from the information available in the metadata. Obviously, semantic error, say an attribute is in the wrong relation, cannot be guessed without understanding the application and thus are out of reach of our automatic analysis. We rather focus on primary and foreign key declarations, or warn if they are missing. The rate of non-null attributes is also checked, with the underlying assumption from our experience that most data are mandatory in a relation. We also check the number of attributes so as to detect a possible insufficient conception effort.

The 13 **style** advices focus on relation and attribute names. Whether a name is significant in the context cannot be checked, so we simply look at their length. Short names are discouraged as they would rather be used as aliases in

queries, with the exception of `id` and `pk` which are accepted as attributes. We also check that the same name does not represent differently typed data, to avoid confusing the user.

The 6 **consistency** advices checks for type and schema consistency in a project, such as type mismatches between a foreign key and the referenced key. As databases may also implement some of these checks, it is possible that some cases cannot be triggered.

The 4 **version** advices focus on database-specific checks, such as capabilities and transaction support, as well as homogeneous choices of back-end engines in a project. This category could also check the actual version of a database used looking for known bugs or obsolescence. Only MySQL-specific version advices are currently implemented.

Finally, the 5 **system** advices, some of which PostgreSQL-specific, check for weak passwords, and key and index issues.

These advices aim at helping the schema developer to improve its relational design. We also use them in our survey to grade projects with a mark from 0 to 10, computed by removing points each time an advice is raised, taking more points if the severity is high, and flooring the result to avoid negative grades. The grading process is normalized using the number of possible occurrences, so that larger projects do not receive lower marks just because of the likelihood of having more issues for their size. Also, points are not removed twice for the same issue: for instance, if a project does not have a single foreign key, the same issue will not be raised again on every tables. Advices not relevant to our open-source database schema survey, *e.g.*, weak password checks, were deactivated.

C. Survey statistical validation

The data collected suggest the influence of some parameters on others. These results deal with general facts about the projects (say foreign keys are more often used with PostgreSQL) or about their grading (say MySQL projects get lower marks). In order to determine significant influences, we applied Pearson's chi-square tests [72] to compute probabilistic degrees of certainty. Beware that these statistical validations hold for our data set only. It is possible that some unwanted bias in the project selection process makes statements that are in reality false appear true, and vice versa. We followed a one project one vote principle in our analyses, so that these validations do not take into account the projects sizes or popularity. Also, our software, as all software, may include bugs with unexpected consequences. Each checked assertion is labeled with an expression indicating the degree of certainty of the influence of one parameter on an other:

very sure The probability is 1% or less to get a result as or more remote from the average. Thus we conclude that there is an influence, with a very high degree of certainty.

rather sure The probability of getting such a result is between 1% and 5% (the usual statistical threshold). Thus there is an influence, with a high degree of certainty.

marginally sure The probability is between 5% and 25%: such a result may have been obtained even if there is no influence. The statement must be taken with a pinch of salt.

not sure The probability is over 25%, or there is not enough available data to compute it. The test cannot assert that there

is a significant influence. Obviously, no such assertion was included in this survey.

The rational for choosing Pearson's chi-square test is that it does not make any assumption about the distribution of values. However, it is crude, and possibly interesting and somehow true results may not be validated. Moreover, the test requires a minimal population, which is not easily reached on our small data set especially when criteria are crossed. Finally, it needs to define distinct populations: for grades or sizes, these populations are cut at the median value in order to perform the test on balanced partitions.

We also computed a correlation matrix to look for possible inter-parameter influence. The result suggested that the parameters are pretty independent beyond the obvious links (say the use of a non-transactional back-end is correlated with isolated tables), and did no help uncover significant new facts.

III. PROJECTS

We discuss the projects considered in this study, grouped by categories, technologies, sizes and release dates. We first present how projects were selected, and then an overview.

A. Project selection

We have downloaded 512 open-source projects starting in the first semester of 2008, adding to our comparison about every project that uses either MySQL [73] or PostgreSQL [74] that we could find and install with reasonable time and effort. The database schemas included in this study are derived from a dump of the database after installation, or from the creation statements when found in the sources. These projects were discovered from various sources: lists and comparisons of software on Wikipedia (Software lists about: photo galleries, content management systems, Internet forums, reference management, issue tracking systems, wikis, social networking, church management, student information systems, accounting, weblog, Internet relay chat, health-care, genealogy, etc.) and other sites; package dependencies from Linux distributions such as Debian [75] or Ubuntu [76] requiring databases; security advisories mentioning SQL [77]; searches on SourceForge [78] which use SQL databases.

Some projects were fixed manually because of various issues, such as: the handling of double-dash comments by MySQL, attribute names (*e.g.*, `out`) rejected by MySQL, bad foreign key declarations or other incompatibilities detected when the projects were forced to use the InnoDB back-end instead of MyISAM, or even some PostgreSQL table definitions including a MySQL specific syntax that were clearly never tested. A particular pitfall of PostgreSQL is that by default syntax errors in statements from an SQL script are ignored and the interpreter simply jumps to the next statement. When installing a project, the flow of warnings often hides these errors. Turning off this feature requires modifying the script, as no command option disables it. More than a dozen PostgreSQL projects contained this kind of issues, which resulted in missing tables or ignored constraint declarations.

Project category	Total		MySQL		PgSQL		Both		Tables		Atts/table	
	nb	%	nb	%	nb	%	nb	%	avg	med	avg	med
CMS	83	16.2	71	18.4	1	3.3	11	11.5	36.6	23	6.6	6.7
System	48	9.4	26	6.7	1	3.3	21	21.9	25.2	9	10.9	7.1
Project	28	5.5	15	3.9	5	16.7	8	8.3	25.4	19	6.9	7.0
Blog	27	5.3	22	5.7	0	0.0	5	5.2	26.8	21	6.9	6.8
Market	22	4.3	21	5.4	0	0.0	1	1.0	53.0	28	7.6	7.2
Forum	19	3.7	17	4.4	0	0.0	2	2.1	23.1	19	8.3	8.6
Accounting	18	3.5	11	2.8	6	20.0	1	1.0	87.8	45	8.8	8.8
Game	16	3.1	16	4.1	0	0.0	0	0.0	26.4	22	6.6	6.9
Mail	16	3.1	8	2.1	1	3.3	7	7.3	10.1	6	5.4	5.0
IRC	13	2.5	6	1.6	1	3.3	6	6.3	14.3	15	6.8	5.8
Homepage	12	2.3	11	2.8	0	0.0	1	1.0	5.1	4	7.0	7.0
Healthcare	11	2.1	6	1.6	2	6.7	3	3.1	89.5	71	11.5	9.5
Phone	11	2.1	5	1.3	2	6.7	4	4.2	18.2	9	14.6	9.0
Address	10	2.0	10	2.6	0	0.0	0	0.0	7.7	7	7.7	7.9
Genealogy	10	2.0	8	2.1	1	3.3	1	1.0	16.4	12	8.4	8.6
Photo	10	2.0	9	2.3	0	0.0	1	1.0	20.2	16	7.1	7.3
Community	9	1.8	7	1.8	0	0.0	2	2.1	17.3	12	8.1	8.0
Music	9	1.8	8	2.1	1	3.3	0	0.0	16.7	8	5.0	6.0
P2P	9	1.8	8	2.1	0	0.0	1	1.0	11.9	7	7.0	8.0
Reference	9	1.8	8	2.1	0	0.0	1	1.0	15.8	16	11.7	8.0
Wiki	9	1.8	7	1.8	1	3.3	1	1.0	15.7	9	5.6	5.7
Calendar	8	1.6	7	1.8	1	3.3	0	0.0	11.1	8	6.1	6.8
Advert	7	1.4	7	1.8	0	0.0	0	0.0	4.0	2	9.0	8.4
Search	6	1.2	6	1.6	0	0.0	0	0.0	18.0	20	6.0	6.0
Student	6	1.2	6	1.6	0	0.0	0	0.0	35.5	28	6.5	6.7
Teaching	6	1.2	3	0.8	1	3.3	2	2.1	13.5	5	4.9	5.3
Conference	5	1.0	4	1.0	1	3.3	0	0.0	73.8	32	6.8	6.2
FAQ	5	1.0	3	0.8	0	0.0	2	2.1	25.0	30	6.6	5.3
Library	5	1.0	4	1.0	1	3.3	0	0.0	63.8	72	7.2	7.3
Survey	5	1.0	3	0.8	0	0.0	2	2.1	25.0	18	6.4	6.4

TABLE I
MAIN CATEGORIES OF PROJECTS, WITH COUNTS, DATABASE SUPPORT AND SIZES

Project technology	Total		MySQL		PgSQL		Both		Tables		Atts/table	
	nb	%	nb	%	nb	%	nb	%	avg	med	avg	med
PHP	399	77.9	335	86.8	8	26.7	56	58.3	29.3	16	7.4	7.2
C	38	7.4	12	3.1	5	16.7	21	21.9	21.3	9	11.5	8.3
Java	22	4.3	8	2.1	6	20.0	8	8.3	67.5	23	9.3	8.2
Perl	21	4.1	10	2.6	5	16.7	6	6.3	44.0	29	6.7	6.7
SQL	8	1.6	6	1.6	1	3.3	1	1.0	27.3	11	4.9	5.0
C++	7	1.4	5	1.3	1	3.3	1	1.0	11.4	6	15.3	6.0
Python	7	1.4	4	1.0	2	6.7	1	1.0	42.9	17	6.5	6.2
Ruby	7	1.4	4	1.0	2	6.7	1	1.0	49.5	16	7.4	6.7

TABLE II
MAIN TECHNOLOGIES OF PROJECTS, WITH COUNTS, DATABASE SUPPORT AND SIZES

B. Overview of projects

We have studied the relational schemas of 512 (see appendix for the full list) open-source projects based on databases: 482 of these run with MySQL, 126 with PostgreSQL, including 96 on both. A project supporting PostgreSQL is very likely to support also MySQL (76%), although the reverse is not true (only 19%) (*very sure*), outlining the relative popularity of these tools. Only 30 projects are PostgreSQL specific. Although there is no deliberate bias in the selection process described in the previous section, where we aimed at completeness, some implicit bias remain nevertheless: for instance, as we can speak mostly English and French, we found mostly international projects advertised in these tongues; Table I shows main project categories, from the personal mundane (game, homepage) to the professional serious (health-care, accounting,

system). Table II shows the same for project technologies. Projects in rare categories or using rare technologies do not appear in these cut-off tables. The result is heavily slanted towards PHP web applications (77%), which seems to reflect the current trend of open-source programming as far as the number of projects is concerned, without indication of popularity or quality. The ratio of PHP projects increases from PostgreSQL only support (26%) to both database support (58%) (*very sure*) to MySQL only support (86%) (*very sure*): PHP users tend to choose specifically MySQL, possibly because of traditional LAMP (*Linux, Apache, MySQL, PHP*) setups advertised with PHP programming. For instance, a search on the Amazon website in January 2012 returns 18 times more results with *PHP MySQL* compared to *PHP PostgreSQL*.

The survey covers 18993 tables (MySQL 13494, Post-

Advice	Lvl.	Cat.	Sev.	MySQL				PostgreSQL			
				Proj	%	Adv	%	Proj	%	Adv	%
Schema without any FK	sch.	design	error	425	88	425	88	70	55	70	55
Tables without PK nor Unique	table	design	error	262	54	1521	11	76	60	1010	18
FK type mismatch	table	consist.	error	2	0	17	0	10	7	153	2
Backend engine inconsistency	sch.	version	error	30	6	30	6	0	0	0	0
FK length mismatch	table	consist.	error	4	0	6	0	2	1	10	0
Integer PK but no other key	table	design	warn	437	90	7470	55	106	84	2509	45
Homonymous heterogeneous attributes	att.	style	warn	296	61	2294	2	76	60	573	1
Unsafe backend engine used in schema	sch.	version	warn	433	89	433	89	0	0	0	0
Attribute count per table over 40	table	design	warn	98	20	220	1	25	19	91	1
Isolated Tables	table	design	warn	30	6	979	7	40	31	1300	23
Tables without PK but with Unique	table	design	warn	117	24	405	3	15	11	40	0
Unique nullable attributes	att.	design	warn	73	15	261	0	23	18	172	0
Nullable attribute rate over 80%	sch.	design	warn	34	7	34	7	25	19	25	19
Redundant indexes	table	system	warn	0	0	0	0	23	18	196	3
Attribute name length too short	att.	style	warn	27	5	91	0	16	12	51	0
Large PK referenced by a FK	table	design	warn	10	2	118	0	19	15	216	3
Table name length too short	table	style	warn	16	3	23	0	7	5	17	0
Composite Foreign Key	table	design	warn	5	1	19	0	8	6	26	0
FK not referencing a PK	table	design	warn	2	0	16	0	7	5	23	0
Redundant FK	table	system	warn	1	0	1	0	2	1	6	0
Non-integer Primary Key	table	design	note	268	55	2261	16	81	64	1729	31
MySQL is used	base	version	note	482	100	482	100	0	0	0	0
Attribute count per table over 20	table	design	note	230	47	684	5	60	47	421	7
Tables with Composite PK	table	design	note	196	40	1781	13	63	50	703	12
Attribute name length quite short	att.	style	note	201	41	748	0	49	38	244	0
Attribute named after its table	att.	style	note	139	28	3114	2	42	33	5033	9
Table without index	table	system	note	0	0	0	0	60	47	719	13
Nullable attribute rate in 50-80%	sch.	design	note	76	15	76	15	33	26	33	26
Table name length quite short	table	style	note	70	14	102	0	28	22	52	0
Table with a single attribute	table	design	note	74	15	419	3	26	20	91	1
Mixed attribute name styles	table	style	note	115	23	1007	7	1	0	37	0
Mixed table name styles	sch.	style	note	51	10	261	54	8	6	22	17
Attribute name length short	att.	style	info	326	67	2911	2	81	64	1047	2
Unsafe backend engine used on table	table	version	info	433	89	10423	77	0	0	0	0
Nullable attribute rate in 20-50%	sch.	design	info	137	28	137	28	41	32	41	32
Table name length short	table	style	info	136	28	258	1	38	30	81	1

TABLE III
LIST OF RAISED ADVICES AND DETAILED COUNTS ABOUT THE 512 PROJECTS

greSQL 5499) containing 166906 attributes (MySQL 114561, PostgreSQL 52345). The project sizes average at 31.2 tables, median 16 (from 1 to 607), with 2 to 10979 attributes. MySQL projects average at 28 tables, median 15 (from 1 to 466), with 238 attributes (from 2 to 9725), while PostgreSQL projects average 44 tables, median 18 (from 1 to 607), with 415 attributes (from 5 to 10979 attributes). The largest MySQL project is OSCARCMMASTER, and the largest PostgreSQL project is ADEMPIERE. Detailed table counts raise from projects with MySQL only support (average 26.4, median 15), to both databases (average 34.0, median 17) or PostgreSQL only (average 75.5, median 30.5). MySQL-only projects are smaller than other projects (*marginally sure*): more ambitious projects seem to use feature-full but maybe less easy to administrate PostgreSQL. However obvious this assertion would seem, the statistical validation is weak because of the small number of projects with PostgreSQL. MySQL projects that use the InnoDB back-end are much larger than their MyISAM counterpart (*very sure*) and are comparable to projects based on PostgreSQL, with 53 tables on average. The number of attributes per table is comparable although

smaller for MySQL (average 8.5 – median 7.0) with respect to PostgreSQL (average 9.5 – median 6.0).

The per-category tables and attributes-per-table counts shows that *accounting*, *health-care* and *market* projects seem more ambitious than other categories (*marginally sure*). The per-technology analysis counts suggests that *Perl*, *Python* and *Java* projects are larger than those based on other technologies (*marginally sure*).

These projects are mostly recent, at least according to their status at an arbitrary common reference date chosen as March 31, 2009: 310 (60%) were updated in the last year, including 179 (34%) in the last six months, and the others are either obsolete or stable. The rate of recently updated projects raises from MySQL-only projects (55%) to projects with both support (73%) (*very sure*) or with PostgreSQL support at (76%) (*very sure*), but there is no significant difference on the recent maintenance figures between projects that are PostgreSQL-only and projects with both databases support.

New data about the status of projects were collected on January 9, 2012. We could not find 69 projects in this new survey (61 MySQL-only, 1 PostgreSQL-only and 7 with both support). Moreover, 153 projects are stale, that is not

updated between the 2009 and 2012 data (128 MySQL-only, 6 PostgreSQL-only and 19 with both support). Nearly half of the MySQL projects are stale or lost, while it is only one quarter of the PostgreSQL projects. MySQL-only projects are more often lost or stale than others in 2012 (*very sure*), and it is still true for MySQL projects compared to PostgreSQL-only projects (*rather sure*). More generally, on these new data, MySQL-only projects are less maintained than others (*very sure*), and it is still true compared to projects with both support (*very sure*) and compared to projects with PostgreSQL-only support (*rather sure*). There are about six months (180 days) between the median update date of MySQL-only projects and PostgreSQL-only projects. Even if we ignore lost and stale projects to focus on projects that were indeed updated in the 2012 data, PostgreSQL-only projects were more recently updated than others (*rather sure*). Yet again, there is no significant update status difference between projects with PostgreSQL support and projects that support both databases on the 2012 data. To conclude, the maintenance of PostgreSQL projects seems more intense: projects that include PostgreSQL support were updated more recently both in 2009 and in 2012.

IV. SURVEY RESULTS

We now analyze the open-source projects of our survey by commenting actual results on MySQL and PostgreSQL, before comparing them. Table III summarizes the advices raised for MySQL and PostgreSQL applications. The first four columns give the advice title, level, category and severity. Then four columns for each database list the results. The first two columns hold the number of projects (*i.e.* schema) tagged and the overall rate. The last two columns give the actual number of advices and rate, which varies depending on the level. A per-project aggregate is also available online [71].

A. Primary keys

A majority of MySQL projects (262 – 54%) have at least one table without neither a primary key nor a unique constraint, and this is even worse with PostgreSQL projects (76 – 60%). The certainty of the observation (*rather sure*) on MySQL-only vs PostgreSQL-only is low because of the small number of projects using the later. As 11% of all MySQL tables and 18% of all PostgreSQL tables do not have any key, the view of relations as sets is hindered as tuples are not identified, and data may be replicated without noticing.

A further analysis gives some more insight. For MySQL, 41% of tables without key do have some KEY option for indexes, but without the UNIQUE or PRIMARY keyword that makes it a key. Having KEY not always declaring a key was clearly a bad design choice. A little 5% of tables without key have an *auto increment* attribute, which suggest uniqueness in practice, but is not enforced. Also, the missing key declaration often seems to be composite. Some tables without key declarations are intended as one tuple only, say to check for the version of the schema or configuration of the application. Similarly, 28% of PostgreSQL tables without key have an index declared. Moreover, 22% have a SERIAL (auto incremented) attribute: Many designers seem to assume

wrongly that SERIAL implies a key. A comment found in the SQLGREY project source suggests that some keys are not declared because of MySQL key size limits.

A simple integer primary key is provided on 61% of tables, with a significantly decreasing rate from MySQL-only (65%) to both database support (62%) (*rather sure*) down to PostgreSQL-only support (39%) (*very sure*). If these primary keys were non-semantic numbers to identify tuples, one would expect at least one other key declared on each table to identify the underlying semantic key. However it is not the case: most (85%) of these tables do not have any other key. When a non simple primary key is available, it is either based on another type or a composite key. The composite keys are hardly referenced, but as the foreign keys are rarely declared one cannot be sure, as shown in the next section.

B. Referential integrity

Foreign keys are important for ensuring data consistency in relational databases. They are supported by PostgreSQL, and by MySQL but with some back-end engines only. In particular, the default MyISAM back-end does not support foreign keys, and this feature was deemed noxious in previous documentations: Version 3.23 includes a *Reasons NOT to Use Foreign Keys constraints* Section arguing that they are only useful to display diagrams, hard to implement and terrible for performance. Foreign key constraints are introduced with the InnoDB engine starting with *MySQL 3.23.44* in January 2001. Although the constraints are ignored by the default MyISAM engine, the syntax is parsed, and triggers the creation of indexes. Version 5.1 documentation has a *Foreign Keys* Section praising the feature, as it *offers benefits*, although it slows down the application. Caveats describe the inconsistencies that may result from *not* using transactions and referential integrity. From a pedagogical perspective, this is a progress.

Foreign key constraints have long been a missing or avoided feature in MySQL and this seems to have retained momentum in many projects, as it is not supported by the default engine: few MySQL projects (57 – 11% of all projects, but 72% of those with InnoDB) use foreign key constraints. The foreign key usage rate is slightly higher (20%) when considering projects supporting both databases (*marginally sure*).

Among MySQL projects, 403 (83%) use only the default MyISAM back-end engine, thus do not have any foreign key checks enabled. In the remainder, 49 (10%) use only InnoDB, and 30 (6%) use a combination of both. More projects (21 – 21%) rely on InnoDB among those supporting both MySQL and PostgreSQL (*marginally sure*). A third of InnoDB projects (30 – 37%) are not consistent in their engine choice: 34% of tables use MyISAM among the 79 InnoDB projects. A legitimate reason for using MyISAM tables in an InnoDB project is that full-text indexes are only available with the former engine. However, this only applies to 11 tables in 6 projects, all other 1441 MyISAM tables in InnoDB projects are not justified by this. A project may decide to store transient data in an unsafe engine (*e.g.*, memory) for performance reason. However, this case is rare, as it represents only 15 tables in 8 projects. About 26% of tables use MyISAM as a default implicit choice in

InnoDB projects, similar to 28% when considering all MySQL projects. Some engine inconsistencies seems due to forgotten declarations falling back to the default MyISAM engine.

We have forced the InnoDB back-end engine for all MySQL projects: 22 additional projects declare 92 new foreign key constraints previously ignored. These new foreign keys are very partial, targeting only some tables. They allow to uncover about two dozen issues, either because the foreign key declaration were failing (say from type errors detected by MySQL) or thanks to analyses from our tool. Additional checks based on foreign keys cannot be raised on schemas that do not declare any of them. Thus *isolated tables* warnings must be compared to the number of projects that do use referential constraints: 30 – 52% of these seem to have forgotten at least some foreign keys, and it is actually the case by checking some of these projects manually.

The foreign key usage is better with PostgreSQL projects, although it is still a minority (56 projects – 44%). This rate is close to the foreign key usage of MySQL projects when considering InnoDB projects only. It gives a better opportunity for additional advices to be checked. The foreign key usage rate raises significantly to 74% when considering PostgreSQL-only projects vs dual support projects (*very sure*).

On the very few projects with partial foreign key declarations, several of these declaration reveal latent bugs, including type mismatch, typically CHAR targeting a VARCHAR or vice versa, or different integers, and type length mismatch, usually non matching VARCHAR sizes. We found 23 such bugs out of the small 1979 declared MySQL attribute constraints, and 163 among the 4424 PostgreSQL constraints. The rate is greater for PostgreSQL, possibly helped by the use of SERIAL which may be considered as a primary key by developers without being declared as such. There are also 153 important warnings related to foreign keys raised for MySQL, and 265 for PostgreSQL. If this error ratio is extrapolated to the number of tables, hundreds of additional latent bugs could be detected using the missing referential constraints.

C. Miscellaneous issues

More issues were found about style, attribute constraints and by comparing projects with dual database support.

There is 13669 noticeable style issues raised from our analyses (7640 for MySQL, 6029 for PostgreSQL), relating to table or attribute names, including a number of one-letter attribute names or two-letters table names. The *id* attribute name is used in the SLASH project with up to 6 different types, mixing various integers and fixed or variable length text types. In PHPETITION, a *date* attribute has types DATE, DATETIME or VARCHAR. 81% of MySQL projects and 78% of PostgreSQL have such style issues.

Many projects do not bother with NOT NULL attribute declarations: 110 MySQL projects (22%) and 58 PostgreSQL projects (46%) have over half of their attributes null-able. This does not reflect the overall use of constraints: for MySQL, the average number of key-related constraints per table is 1.07 (from BOARDPLUS 0.00 to JWHOISERVER 3.57), while for PostgreSQL it is 1.24 (from ANDROMEDA 0.00 to ADEMPIERE

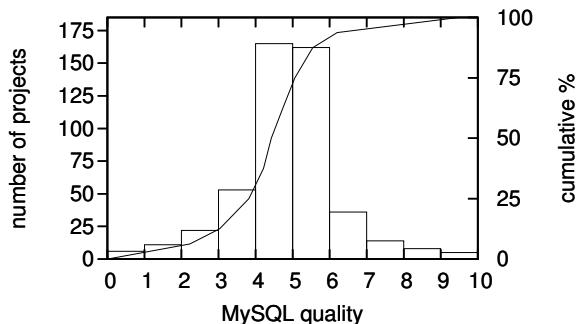
4.25). Project ANDROMEDA is astonishing: there is not a single constraint declared (no primary key, no foreign key, no unique, no not null) on the 180 tables, although there are a number of non-unique indexes and of sequences.

It is interesting to compare the schemas of the 96 projects available with both databases. This dual support must not be taken at face value: PostgreSQL support is often an afterthought and is not necessarily functional, including project such as ELGG, TAGADASH, QUICKTEAM or TIKIWIKI where some PostgreSQL table declarations use an incompatible MySQL syntax; 38 (39%) projects have missing tables or attributes between the MySQL and PostgreSQL versions: 398 tables and 191 individual attributes are missing or misspelled one side or another. Among the missing tables, 73 look like some kind of sequence, and thus might be possibly legitimate, although why the *auto increment* feature was not satisfactory is unclear. At the minimum, the functionalities are not the same between the MySQL and PostgreSQL versions of these projects.

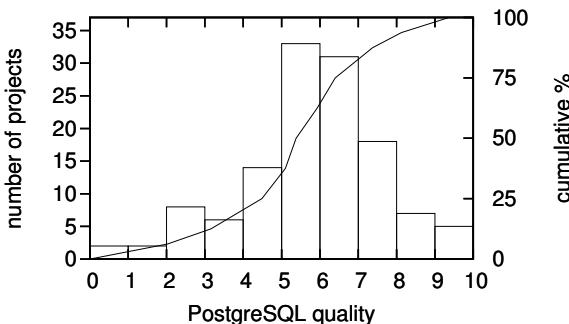
D. Overall quality

We have computed a synthetic project quality evaluation ranging from 10 (good) to 0 (bad) by removing points based on advice severity (error, warning, notice), level (schema, table, attribute) and project size. The MySQL projects quality average is 4.4 ± 1.4 (from 9.5 JWHOISERVER to 0.0 MANTIS), significantly lower than PostgreSQL 5.4 ± 1.8 (from 9.4 COMICS to 0.0 NURPAWIKI) (*very sure*). This does not come as a surprise: most MySQL projects choose the default data-unsafe MyISAM engine, hence incur a penalty. Also, the multiplicity of MySQL back-ends allows the user to mix them unintentionally, what is not possible with PostgreSQL. When all MySQL-specific advices are removed, the quality measure is about the same for both databases. However, as PostgreSQL schemas provide more information about referential integrity constraints, they are also penalized as more advices can be raised based on the provided additional information. For projects which support both databases, the grade's correlation is significant and positive (0.55), which is logical as the same style warnings are triggered on both sides.

Table IV shows the projects per quality decile. The PostgreSQL-only project quality is more spread than MySQL projects (*very sure*). Table V compares the quality of projects according to size, with small up to 9, medium up to 29, and large otherwise. The quality is quite evenly distributed among sizes, which suggests that our effort to devise a size-neutral grading succeeded. Table VI compares quality based on the project categories. The number of projects in each category is too small to draw deep conclusions. Table VII addresses the technology used in the project: Java and Python lead while C, PHP and Ruby are near bottom. PHP projects take less care of their relational design (*rather sure*), but this may be explained by the fact that MySQL is used more often in these projects, and that an unsafe engine is selected more often (*very sure*). Yet again, the very small count of projects with some of the technologies do not allow to draw deep conclusion about them. Finally, Table VIII and Table IX show that quality evaluation does not change much depending whether projects are updated more often.

TABLE IV
QUALITY PER DECILE

Size	MySQL projects						
	nb	%	avg	σ	min	med	max
small	181	38	4.7 ± 1.4	0.0	4.5	9.5	
medium	164	34	4.2 ± 1.3	0.0	4.3	8.7	
large	137	28	4.3 ± 1.4	0.0	4.4	8.2	

TABLE IV
QUALITY PER DECILE

Size	PostgreSQL projects						
	nb	%	avg	σ	min	med	max
small	44	35	5.3 ± 2.0	0.0	5.3	9.4	
medium	37	29	5.5 ± 1.5	2.0	5.3	9.3	
large	45	36	5.3 ± 2.0	0.0	5.7	8.1	

TABLE V
QUALITY PER SIZE

Category	MySQL projects						
	nb	%	avg	σ	min	med	max
irc	12	2	5.1 ± 1.3	2.0	5.4	7.0	
mail	15	3	4.4 ± 1.7	1.7	4.7	8.4	
project	23	5	4.3 ± 1.4	0.0	4.6	6.2	
system	47	10	4.5 ± 1.4	0.0	4.5	9.5	
game	16	3	4.4 ± 2.0	0.9	4.5	9.1	
blog	27	6	4.4 ± 0.9	2.5	4.5	7.2	
forum	19	4	4.3 ± 0.9	2.4	4.4	5.7	
cms	82	17	4.2 ± 1.1	0.0	4.3	8.3	
homepage	12	2	4.1 ± 0.9	3.0	4.1	5.9	
market	22	5	4.0 ± 1.4	1.8	4.0	8.2	
accounting	12	2	4.4 ± 1.9	1.9	3.6	7.5	

TABLE VI
QUALITY PER PROJECT MAIN CATEGORIES

Category	PostgreSQL projects						
	nb	%	avg	σ	min	med	max
teaching	3	2	7.9 ± 2.2	5.3	8.9	9.4	
blog	5	4	6.6 ± 1.1	5.3	6.4	8.2	
accounting	7	6	5.9 ± 2.0	2.0	6.4	7.8	
cms	12	10	6.1 ± 1.3	4.0	5.9	8.1	
irc	7	6	5.4 ± 1.7	2.0	5.6	7.4	
phone	6	5	5.2 ± 1.5	3.1	5.3	7.4	
project	13	10	5.4 ± 1.6	2.2	5.2	9.3	
system	22	17	5.0 ± 2.1	1.6	5.1	9.0	
mail	8	6	4.9 ± 1.6	3.0	4.8	7.5	
healthcare	5	4	3.2 ± 2.7	0.0	3.3	6.6	

TABLE VII
QUALITY PER PROJECT MAIN TECHNOLOGIES

Techno.	MySQL projects						
	nb	%	avg	σ	min	med	max
python	5	1	5.9 ± 2.0	3.7	6.2	8.2	
sql	7	1	4.0 ± 2.5	0.0	5.3	5.9	
java	16	3	4.8 ± 2.8	0.0	5.2	9.5	
c++	6	1	4.8 ± 1.2	3.3	4.5	7.0	
c	33	7	4.6 ± 1.4	2.0	4.4	8.4	
php	391	81	4.4 ± 1.2	0.0	4.4	9.1	
perl	16	3	3.9 ± 2.1	0.0	4.3	8.7	
ruby	5	1	4.5 ± 0.9	3.7	4.2	5.6	

Techno.	PostgreSQL projects						
	nb	%	avg	σ	min	med	max
python	3	2	7.0 ± 0.6	6.6	6.8	7.7	
java	14	11	6.1 ± 2.4	0.0	6.8	9.3	
c++	2	2	6.7 ± 1.0	6.0	6.7	7.4	
perl	11	9	6.0 ± 1.9	2.0	6.1	8.9	
sql	2	2	5.8 ± 5.1	2.2	5.8	9.4	
php	64	51	5.2 ± 1.6	0.0	5.4	8.2	
ruby	3	2	5.1 ± 1.2	4.0	5.0	6.3	
c	26	21	4.8 ± 1.9	1.6	5.0	9.0	

Date	MySQL projects						
	nb	%	avg	σ	min	med	max
recent	162	34	4.3 ± 1.3	0.0	4.4	8.6	
older	320	66	4.4 ± 1.4	0.0	4.4	9.5	

TABLE VIII
QUALITY PER PROJECT UPDATE IN MARCH 2009

Date	PostgreSQL projects						
	nb	%	avg	σ	min	med	max
recent	59	47	5.3 ± 1.6	0.0	5.3	9.3	
older	67	53	5.4 ± 2.0	0.0	5.6	9.4	

Date	MySQL projects						
	nb	%	avg	σ	min	med	max
recent	112	23	4.3 ± 1.3	0.0	4.5	7.8	
older	155	32	4.4 ± 1.5	0.0	4.5	9.5	
stale	147	30	4.5 ± 1.4	0.9	4.4	9.1	
lost	68	14	4.2 ± 1.0	0.0	4.2	6.3	

TABLE IX
QUALITY PER PROJECT UPDATE IN JANUARY 2012

Date	PostgreSQL projects						
	nb	%	avg	σ	min	med	max
recent	41	33	5.7 ± 1.4	0.0	5.6	7.7	
older	52	41	5.2 ± 1.9	0.0	5.3	9.3	
stale	25	20	5.1 ± 2.2	0.7	5.3	9.4	
lost	8	6	5.5 ± 2.3	2.0	5.5	9.0	

V. CONCLUSION

This is the first survey on the quality of relational schemas in open-source software. The overall quality results are worse than envisioned at the beginning of the study. Although we did not expect a lot of perfect projects, having so few key declarations and referential integrity constraints came as a surprise. We must acknowledge that our assumption that data are precious, and that the database should help preserve its consistency by enforcing integrity constraints and implementing transactions, is not shared by most open-source projects, especially when based on MySQL and PHP. This is illustrated by bug report 15441 [79] about missing keys on tables in MEDIAWIKI, the software behind Wikipedia: it had no effect on the software after more than three years, although it triggered some discussions at the beginning of 2012.

We can only speculate about the actual reasons that explain the poor quality of the surveyed schemas in open-source projects. One way to investigate further these issues would be to collect data about and from the people who designed the relational schemas of these projects. For instance, if MySQL or PHP users are found less savvy about software development, that could account for a lower quality and maintenance of the corresponding projects. Some interesting questions could be investigated: What are their educational and professional background? Did they receive any formal education about computer programming in general? About relational database design in particular? Do they consider database design as an important issue? How are they perceiving the actual quality of their schemas, and the quality of their software? When did they started database design? For MySQL, what database engines do they use? Did the initial policy of discouraging foreign key usage influence them? We attempted to conduct such a survey by contacting some people by e-mail and encouraging them to fill a web form online. The return ratio of this survey attempt was *null*. This establishes the fact that schema designers in open-source software do not wish to answer such questions, with a very high degree of accuracy.

Another relevant question is whether our results would be different if we studied closed-source projects developed by payed professionals, possibly using non open-source database technologies from Oracle or Microsoft. However, accessing such data at a level compatible with statistical validation seems very difficult. If we were to believe some of our experience, the results could end up being quite similar, especially when considering PHP/MySQL projects.

It is interesting to note that the first author contributed both to the best PostgreSQL project (COMICS), and to one of the worst MySQL project (SLXBBL), which is *Salix* executed on its own schema. This deserves an explanation: COMICS is a small database used for teaching SQL. The normalized schema emphasizes clarity and cleanliness with a pedagogic goal in mind. Even so, the two raised warnings deserve to be fixed, although one would require an additional attribute. SLXBBL tables generate a lot of errors, because they are views materialized for performance issues. Also, they rely on MyISAM because some SQL create table statements must be compatible with both MySQL and PostgreSQL to ease the tool

portability. Nevertheless, the comparison of schemas allowed to find one bug: an attribute had a different name, possibly because of a bad copy-paste.

Acknowledgement

We are indebted to Pierre Jouvelot for helping with the title and proof reading. We also thank the anonymous reviewers for their helpful remarks that we tried to address for the better of the paper.

REFERENCES

- [1] F. Coelho, A. Ailos, S. Pilot, and S. Valeev, "A Field Analysis of Relational Database Schemas in Open-source Software," in *DBKDA: 3rd Int. Conf. on Advances in Databases, Knowledge, and Data Applications*, IARIA, Ed., no. ISBN:978-1-61208-002-4, St Marteen, The Netherlands Antilles, Jan. 2011, pp. 9–15.
- [2] J. M. Gonzales-Barahona, P. Heras Quiros, and T. Bollinger, "A brief history of free software and open source," *IEEE Software*, pp. 32–33, Jan. 1999.
- [3] R. Stallman, "GNU Project announcement," <http://www.gnu.org/gnu/initial-announcement.html> (2012-01-06), Sep. 1983.
- [4] ———, "FSF: Free Software Foundation," Oct. 1985, www.fsf.org, (2012-01-06).
- [5] A. Deshpande and D. Riehle, "The Total Growth of Open Source," in *4th Conf. on Open Source Systems (OSS)*. Springer Verlag, 2008, pp. 197–209.
- [6] L. F. Wurster, "As Number of Business Processes Using Open-Source Software Increases, Companies Must Adopt and Enforce an OSS Policy," Gartner Inc, Sep. 2008, iD Number: G00160997.
- [7] D. C. Plummer, B. Gammage, K. Harris-Ferrante, and J. Lopez, "Predicts 2010: Revised Expectations for IT Demand, Supply and Oversight," Gartner, Inc, Dec. 2009, iD Number: G00173560.
- [8] "Open Source Licences," <http://opensource.org> (2012-01-06), Feb. 1998.
- [9] K. Crowston, H. Annabi, and J. Howison, "Defining open source software project success," in *24th Int. Conf. on Information Systems (ICIS)*, 2003, pp. 327–340.
- [10] S. Görling, "A critical approach to open source software," <http://flosshub.org/196> (2012-01-06), 2003.
- [11] C. Gacek, T. Lawrie, and B. Arief, "The many meanings of open source," *IEEE Software*, vol. 21, pp. 34–40, 2004.
- [12] E. von Hippel, B. Mako Hill, and K. Lakhani, "Free and opensource software research community," <http://opensource.mit.edu>, now offline, Nov. 2001.
- [13] A. Hars, "Working for free? motivations for participating in open-source projects," *Int. J. of Electronic Commerce*, vol. 6, pp. 25–39, 2002, also IEEE 34th Hawaii Int. Conf. on System Sciences 2001.
- [14] G. Hertel, S. Niedner, and S. Herrmann, "Motivation of software developers in open source projects: An internet-based survey of contributors to the linux kernel," *Research Policy*, vol. 32, pp. 1159–1177, 2003.
- [15] I. horn Hann, J. Roberts, S. Slaughter, and R. Fielding, "An empirical analysis of economic returns to open source participation (unpublished working paper)," 2004.
- [16] A. Bonacorsi and C. Rossi, "Altruistic individuals, selfish firms? the structure of motivation in open source software," Santa Anna School of Advanced Studies. Institute for Informatics and Telematics, Tech. Rep., Jan. 2004, First Monday, <http://firstmonday.org/> (2012-01-06).
- [17] K. J. Stewart and S. Gosain, "The impacts of ideology on effectiveness in open source software development teams (working paper)," *MIS Quarterly*, vol. 30, pp. 291–314, 2005.
- [18] J. E. Cook, "Open source development: An arthurian legend. making sense of the bazaar," in *Proceedings of the 1st Workshop on Open Source Software*, 2001.
- [19] M. S. Elliott and W. Scacchi, "Mobilization of software developers: The free software movement," 2006.
- [20] ———, "Free software: A case study of software development in a virtual organizational culture," in *A Virtual Organizational Culture*, Working Paper. Institute for Software Research, Tech. Rep., 2003.
- [21] M. S. Elliott, "Free software developers as an occupational community: Resolving conflicts and fostering," in *Collaboration, Proc. ACM Int. Conf. Supporting Group Work*, 2003, pp. 21–30.
- [22] K. Healy and A. Schussman, "The ecology of open-source software development," Dept of Sociology, Univ. of Arizona, Tech. Rep., 2003.

- [23] K. Crowston and H. Annabi, "Effective work practices for software engineering: Free/libre open source software development," in *in Proc. of WISER*. ACM Press, 2004, pp. 18–26.
- [24] W. Seidel and C. Niedermeier, "Open source software: Leveraging software quality in the industrial context." OSSIE, 2003.
- [25] W. Scacchi, J. Feller, B. Fitzgerald, S. Hissam, and K. Lakhani, "Understanding Free/Open Source Software Development Processes," *Software Process: Improvement and Practice*, vol. 11, no. 2, pp. 95–105, May 2006.
- [26] J. D. Herbsleb, A. Mockus, T. A. Finholt, and R. E. Grinter, "An empirical study of global software development: Distance and speed," in *In 23rd Int. Conf. on Software Engineering*. IEEE Computer Society, 2001, pp. 81–90.
- [27] B. J. Dempsey, D. Weiss, P. Jones, and J. Greenberg, "A quantitative profile of a community of open source linux developers," University of North Carolina at Chapel Hill, Tech. Rep., 1999.
- [28] R. A. Ghosh, R. Glott, B. Krieger, and G. Robles, "Free/libre and open source software: Survey and study, floss, part 4: Survey of developers," Int. Institute of Infonomics, University of Maastricht, The Netherlands, Tech. Rep., Jun. 2002.
- [29] D. M. Nichols and M. B. Twidale, "The usability of open source software," *First Monday*, vol. 8, 2003.
- [30] Eclipse Foundation, "The open source developer report, 2010 eclipse community survey," Tech. Rep., Jun. 2010.
- [31] J. Lerner and J. Tirole, "The economics of technology sharing: open source and beyond. working paper 10956. retrieved jun 7, 2005 <http://www.nber.org/papers/w10956>," *J. of Economic Perspectives*, vol. 19, pp. 99–120, 2004.
- [32] K. M. Schmidt and M. Schnitzer, "Public subsidies for open source? some economic policy," 2002, cEPR Discussion Paper 3793.
- [33] Netcraft Ltd, "Web Server Survey," <http://news.netcraft.com/> (2012-01-06), 2012, running since 1995.
- [34] A. Mockus, R. T. Fielding, and J. Herbsleb, "Two case studies of open source software development: Apache and mozilla," *ACM Transactions on Software Engineering and Methodology*, vol. 11, pp. 309–346, 2002.
- [35] K. R. Lakhani, "How open source software works: "free" user-to-user assistance," *Research Policy*, pp. 923–943, 2000.
- [36] B. Mishra, A. Prasad, and S. Raghunathan, "Quality and Profits Under Open Source Versus Closed Source," in *ICIS*, no. 32, 2002.
- [37] I. Stamelos, L. Angelis, A. Oikonomou, and G. L. Bleris, "Code quality analysis in open-source software development," *Information Systems J.*, *2nd Special Issue on Open-Source*, vol. 12, no. 1, pp. 43–60, Feb. 2002, blackwell Science.
- [38] E. Capra, C. Francalanci, and F. Merlo, "En Empirical Study on the Relationship among Software Design Quality, Development Effort and Governance in Open Source Projects," *IEEE Software Engineering*, vol. 34, no. 6, pp. 765–782, nov-dec 2008.
- [39] R. Gobeille, "The FOSSology Project," in *Working Conf. on Mining Software Repositories*, no. 5, Leipzig, Germany, May 2008.
- [40] G. Concas, M. Marchesi, A. Murgia, R. Tonelli, and I. Turnu, "On the distribution of bugs in the eclipse system," *IEEE Transactions on Software Engineering*, vol. 99, no. PrePrints, 2011.
- [41] Coverty, "Coverty scan open source report," Coverty, White Paper, 2009.
- [42] Veracode, Inc, "State of security report," White paper, Mar. 2010.
- [43] C. Graham, D. Sommer, and B. Sood, "Market Share: Relational Database Management Systems by Operating System, Worldwide, 2006," Gartner, Inc, Jun. 2007, iD Number: G00149469.
- [44] D. Litchfield, "The Database Exposure Survey 2007," NGSSoftware Insight Security Research (NISR), Nov. 2007.
- [45] E. F. Codd, "A relational model for large shared databanks," *Communications of the ACM*, vol. 13, no. 6, pp. 377–387, Jun. 1970.
- [46] ISO/IEC, "Information technology - database languages - SQL," 2003, standard 9075.
- [47] ISO/IEC, Ed., *9075-11:2003: Information and Definition Schemas (SQL/Schemata)*. ISO/IEC, 2003.
- [48] W. C. Burkett, "Database Schema Design Quality Principles," <http://www.intergate.com/~wcb/DbSchemaQuality.pdf>, (2012-01-08), Dec. 1997.
- [49] O. Herden, "Measuring Quality of Database Schemas by Reviewing – Concept, Criteria and Tool," in *5th Int. ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE 2001)*, Budapest, Hungary, Jun. 2001.
- [50] J. Lemaitre and J.-L. Hainaut, "Transformation-based Framework for the Evaluation and Improvement of Database Schemas," in *Int. Conf. on Advanced Information Systems Engineering (CAiSE)*, Hammamet, Tunisia, Jun. 2010.
- [51] ———, "Quality Evaluation and Improvement Framework for Database Schemas Using Defect Taxonomies," in *Int. Conf. on Advanced Information Systems Engineering (CAiSE)*, London, United Kingdom, Jun. 2011.
- [52] T. J. MacCabe, "A Complexity Measure," *IEEE Software Engineering*, vol. SE-2, no. 4, pp. 308–320, Dec. 1976.
- [53] M. H. Halstead, *Elements of Software Science*. New York, USA: Elsevier, 1977, no. ISBN:0444002057.
- [54] H. F. Li and W. K. Cheung, "An empirical study of software metrics," *IEEE Transactions on Software Engineering*, 1987.
- [55] M. Piattini, M. Genero, C. Calero, and G. Alarcos, "Data model metrics," in *In Handbook of Software Engineering and Knowledge Engineering: Emerging Technologies*, World Scientific, 2002.
- [56] M. Genero, "A survey of Metrics for UML Class Diagrams," *J. of Object Technology*, vol. 4, pp. 59–92, Nov. 2005.
- [57] H. M. Sneed and O. Foshag, "Measuring legacy database structures," in *European Software Measurement Conf. (FESMA'98)*, Hooft and Peeters, Eds., 1998.
- [58] M. Piattini, C. Calero, and M. Genero, "Table Oriented Metrics for Relational Databases," *Software Quality J.*, vol. 9, no. 2, pp. 79–97, 2001.
- [59] A. L. Baroni, C. Calero, F. Ruiz, and F. Brito e Abreu, "Formalizing object-relational structural metrics," in *Conf. of APSI, Lisbon*, no. 5, Nov. 2004.
- [60] C. Calero, M. Piattini, and M. Genero, "Empirical validation of referential integrity metrics," *Information and Software Technology*, vol. 43, no. 15, pp. 949–957, Dec. 2001.
- [61] A. Bessey, K. Block, B. Chelf, A. Chou, B. Fulton, S. Hallem, C. Henri-Gros, A. Kamsky, S. McPeak, and D. Engler, "A Few Billion Lines of Code Later: Using Static Analysis to Find Bugs in the Real World," *Communication of the ACM*, vol. 53, no. 2, pp. 66–75, Feb. 2010.
- [62] A. Cleve, J. Lemaitre, J.-L. Hainaut, C. Mouchet, and J. Henrard, "The role of implicit schema constructs in data quality," in *Workshop on Management of Uncertain Data (MUD)*, Auckland, New Zealand, Aug. 2008, pp. 33–40.
- [63] A. Aillo, S. Pilot, S. Valeev, and F. Coelho, "Salix Babylonica: advices about database relational schemas," Software from <http://coelho.net/salix/> (2012-01-06), Aug. 2008, version 1.0.0 on 2012-01-27.
- [64] F. Coelho, "PG-Advisor: proof of concept SQL script," Mailed to pgsql-hackers, Mar. 2004.
- [65] J. Currier, "SchemaSpy: Graphical database schema metadata browser," Source Forge, Aug. 2005, (2012-01-06).
- [66] B. Schwartz and D. Nichter, "Maatkit," Google Code, 2007, see *duplicate-key-checker* and *schema-advisor*. Part of the Percona Toolkit as of 2012-01-06 (<http://www.percona.com/software/percona-toolkit/>).
- [67] J. Berkus, "Ten ways to wreck your database," O'Reilly Webcast, Jul. 2009, (2012-01-06).
- [68] A. Boehm, M. Wetzka, A. Sickmann, and D. Seipel, "A Tool for Analyzing and Tuning Relational Database Applications: SQL Query Analyzer and Schema EnHancer (SQUASH)," in *Workshop über Grundlagen von Datenbanken*, Jun. 2006, pp. 45–49.
- [69] G. Singh, "PostgreSQL Adviser," Software at http://git.postgresql.org/gitweb/pg_adviser.git (2012-01-06), Jul. 2007.
- [70] E. F. Codd, "Is Your DBMS Really Relational? Does Your DBMS Run By The Rules?" *ComputerWorld*, Oct. 1985.
- [71] F. Coelho, "Database quality survey projects and results," Jan. 2012, detailed list of projects surveyed in *On the Quality of Relational Database Schemas in Open Source Software*, report A/478/CRI. [Online]. Available: <http://www.coelho.net/salix/projects.html>
- [72] K. Pearson, "On the Criterion that a Given System of Deviations from the Probable in the Case of a Correlated System of Variables is such that it Can Reasonably Be Supposed to have Arisen from Random Sampling," *Philosophical magazine*, vol. 5, no. 50, pp. 157–175, Jul-Dec 1900, Taylor & Francis Ed, London.
- [73] MySQL AB, "MySQL – Relational Database Management System," <http://mysql.com/> (2012-01-06), May 1995.
- [74] PostgreSQL Global Development Group, "PostgreSQL – Object-Relational Database Management System," <http://postgresql.org/> (2012-01-06), Aug. 1996, based on the Postgres, which started in 1986.
- [75] "Debian," <http://debian.org/> (2012-01-06), Aug. 1993.
- [76] Canonical Ltd, "Ubuntu," <http://ubuntu.com/> (2012-01-06), Oct. 2004.
- [77] SecurityFocus, "Security advisories," <http://securityfocus.com/> (2012-01-06), Jan. 1999.
- [78] "Source Forge," <http://sourceforge.net/> (2012-01-06), 1999.
- [79] F. Coelho, "MediaWiki bug 15441," https://bugzilla.wikimedia.org/show_bug.cgi?id=15441 (2012-01-06), Sep. 2008.

APPENDIX
LIST OF ADVICES

- 1) **Schema without any FK** *schema design error*
Why use a relational database if data are not related at all?
Well, that might happen...
- 2) **No attribute in table** *table design error*
There must be something in a table.
- 3) **Tables without PK nor Unique** *table design error*
All tuples must be uniquely defined to be consistent with the set theory. There is no unique subset of attribute which can be promoted as a PK.
- 4) **Nullable attribute rate over 80%** *schema design warning*
Warning: Most of the time, attributes should be NOT NULL. Too high a rate of nullable attribute may reveal that some fields are lacking a NOT NULL.
- 5) **Attribute count per table over 40** *table design warning*
Having so many attributes in the same table may reveal the need for additional relations.
- 6) **Composite Foreign Key** *table design warning*
As for primary keys, simple foreign keys are usually better design, and make updates easier.
- 7) **FK not referencing a PK** *table design warning*
A Foreign Key should rather reference a Primary Key.
- 8) **Integer PK but no other key** *table design warning*
A simple integer primary key suggests that some other key must exist in the table.
- 9) **Isolated Tables** *table design warning*
In a database design, tables are usually linked together.
- 10) **Large PK referenced by a FK** *table design warning*
Having large primary keys referenced by a foreign key may reveal data duplication, as the primary key is likely to contain relevant information.
- 11) **Tables without PK but with Unique** *table design warning*
All tables should have a primary key to be consistent with the set theory. A unique constraint may be promoted as the primary key.
- 12) **Attribute has a pseudo 'NULL' text default** *attribute design warning*
Possibly the NULL value was intended instead of the 'NULL' text.
- 13) **Unique nullable attributes** *attribute design warning*
A unique nullable attribute may be a bad design if NULL does not have a particular semantic.
- 14) **Nullable attribute rate in 50-80%** *schema design notice*
Notice: Most of the time, attributes should be NOT NULL. Too high a rate of nullable attribute may reveal that some fields are lacking a NOT NULL.
- 15) **Attribute count per table over 20** *table design notice*
Having many attributes in the same table may suggest the need for additional relations.
- 16) **Non-integer Primary Key** *table design notice*
Having integer primary keys without specific application semantics make updates easier.
- 17) **Table with a single attribute** *table design notice*
Possibly some more attributes are needed to have a semantic.
- 18) **Tables with Composite PK** *table design notice*
A simple primary key, without specific semantics, is usually a better design, and references through foreign keys are simpler.
- 19) **Nullable attribute rate in 20-50%** *schema design information*
Information: Most of the time, attributes should be NOT NULL. Too high a rate of nullable attribute may reveal that some fields are lacking a NOT NULL.
- 20) **FK length mismatch** *table consistency error*
A Foreign Key should have matching referencing and referenced type sizes.
- 21) **FK type mismatch** *table consistency error*
A Foreign Key should have matching referencing and refer-
- enced types.
- 22) **Destination table and FK in different schemas** *table consistency warning*
A constraint and its destination table are usually in the same schema.
- 23) **Source table and constraint in different schemas** *table consistency warning*
A constraint and its source table should be in the same schema.
- 24) **Table and index in different schemas** *table consistency warning*
An index and its table should be in the same schema.
- 25) **Tables linked but in different schemas** *table consistency notice*
Linked tables are usually in the same schema.
- 26) **Backend engine inconsistency** *schema version error*
Different backends are used in the same database. It may be legitimate to do so if a particular feature of one backend is needed, for instance full text indexes.
- 27) **Unsafe backend engine used in schema** *schema version warning*
An unsafe backend (e.g. MyISAM) used at least once lacks referential integrity, transaction support, and is not crash safe.
- 28) **MySQL is used** *database version notice*
MySQL lacks important features of the SQL standard, including missing set operators.
- 29) **Unsafe backend engine used on table** *table version information*
An unsafe backend (e.g. MyISAM) lacks referential integrity, transaction support, and is not crash safe.
- 30) **Schema name length too short** *schema style warning*
A schema name with less than 3 characters is really too short.
- 31) **Table name length too short** *table style warning*
A table name with less than 2 characters is really too short.
- 32) **Attribute name length too short** *attribute style warning*
An attribute name with 1 character is really too short.
- 33) **Homonymous heterogeneous attributes** *attribute style warning*
Better avoid using the same attribute name with different types on different tables in the same application, as it may confuse the developer.
- 34) **Mixed table name styles** *schema style notice*
Better use homogeneous table names.
- 35) **Schema name length quite short** *schema style notice*
A schema name with 4 characters is quite short.
- 36) **Mixed attribute name styles** *table style notice*
Better use homogeneous attribute names.
- 37) **Table name length quite short** *table style notice*
A table name with 3 characters is quite short.
- 38) **Attribute name length quite short** *attribute style notice*
An attribute name of 2 characters is quite short (but "id" and "pk").
- 39) **Attribute named after its table** *attribute style notice*
An attribute contains the name of its table, which is redundant.
- 40) **Schema name length short** *schema style information*
A schema name with 5 characters is short.
- 41) **Table name length short** *table style information*
A table name with 4 characters is short.
- 42) **Attribute name length short** *attribute style information*
An attribute name with 3 characters is short.
- 43) **SuperUser with weak password** *user system error*
SuperUser with empty or username password.
- 44) **Redundant FK** *table system warning*
Redundant Foreign Keys are costly to maintain.
- 45) **Redundant indexes** *table system warning*
Redundant indexes are costly to maintain.
- 46) **User with weak password** *user system warning*
User with empty or username password.
- 47) **Table without index** *table system notice*
Not a single index on a table.

Using Statistical Information for Efficient Design and Evaluation of Hybrid XML Storage

Lena Strömbäck

Swedish Meteorological and Hydrological Institute
Folkborgsvägen 1, 601 76 Norrköping
lena.stromback@smhi.se

Valentina Ivanova, David Hall

Department of Computer and Information Science
Linköpings Universitet
S-581 83 Linköping, Sweden
valentina.ivanova@liu.se, david@dpg.se

Abstract — Modern relational database management systems provide hybrid XML storage, combining relational and native technologies. Hybrid storage offers many design alternatives for XML data. In this paper we explore how to aid the user in effective design of hybrid storage. In particular we investigate how the XML schema and statistical information about the data can support the storage design process. In our previous work, we presented our tool HShreX that uses statistical information about a data set to enable fast evaluation of alternative hybrid design solutions. In this paper, we extend this work by presenting more details about the tool and results of an extended evaluation. In particular, this paper gives a detailed presentation on how the tool aids in the storage design and evaluation process.

Keywords – XML, Hybrid XML management, indexing, storage design.

I. INTRODUCTION

The rapid increase in web based applications yields an increasing interest in using XML (eXtensible Markup Language) for representation of data. XML is able to represent all kinds of data ranging from marked-up text, through so called semi-structured data to traditional, well-structured datasets. Supporting the flexibility that makes XML appealing is challenging from data management and technical perspectives. Several approaches have been used including native databases and shredding XML documents into relations. In practice, hybrid storage that combines native and relational solutions is of large interest. Hybrid storage is provided by the major relational database vendors (Oracle, IBM DB2 and Microsoft SQL Server). They offer interesting options for storage design where native and relational storage can be used side by side. In our previous work [1], we present our tool HShreX that uses statistical information about a data set to aid in hybrid storage design.

Several studies evaluate different solutions for XML management. As an example, [2] and [3] provide general benchmarks for XML data while [4] and [5] gives a case study of XML data within bioinformatics. For shredding, a number of different strategies are available [6]. It is well known that the choice of translation strategy affects the efficiency [7][8][9] and that the translation can be optimized in many ways. However, comparisons of different storage strategies [10] and hybrid XML storage [11] [12] [13] has, so

far, only been studied in a few cases. The above studies discuss a number of features that may have an impact on how to achieve efficient storage; the complexity and regularity of the XML structure; how the data is queried, i.e., the access patterns for different entities in the data set; and the frequency of references to other sources.

In this paper, we further explore these issues by investigating the impact of the application on the performance of the database. The properties we are focusing on are the XML schema structure and statistical properties of the data set. In Section II, we motivate and discuss the goals of our work that extends the discussion from [1]. This is followed by a discussion of properties and measurements relevant for storage design in Section III. We present our tool that enables fast evaluation and exploration of storage solutions in Section IV. Here, we extend the presentation from [1], which give a better understanding on how the tool can be used for a fast analyze of properties for a dataset. Statistical analysis of the data sets used for the tool evaluation is presented in Section V. In Section VI, we further extend the previous evaluation to show the feasibility of the tool. Related work is presented in Section VII. The paper is summarized by presenting our future vision in Section VIII. Our long term goal with the work is to present a method that can suggest a set of plausible hybrid storage models for an application.

II. MOTIVATION AND GOALS

Previous work [5][7][14][15][16] has defined efficient shredding methods for XML data into relational databases that result in fast query times. For hybrid storage, the situation is more complex where an inappropriate choice of storage design can lead to poor performance [17]. In general, automatically shredded relational XML mappings can lead to a rather large and complicated structure of relations. On the other hand, storing entire XML documents natively in XML keeps the structure completely intact to the cost of slow access to the data. For hybrid XML storage, we have the choice to store parts of the XML structure as relations and other parts as XML and can gain from the benefit of a good data model and relatively fast performance. The design of a good hybrid storage model is complex and dependent on the requirements for the specific application [17].

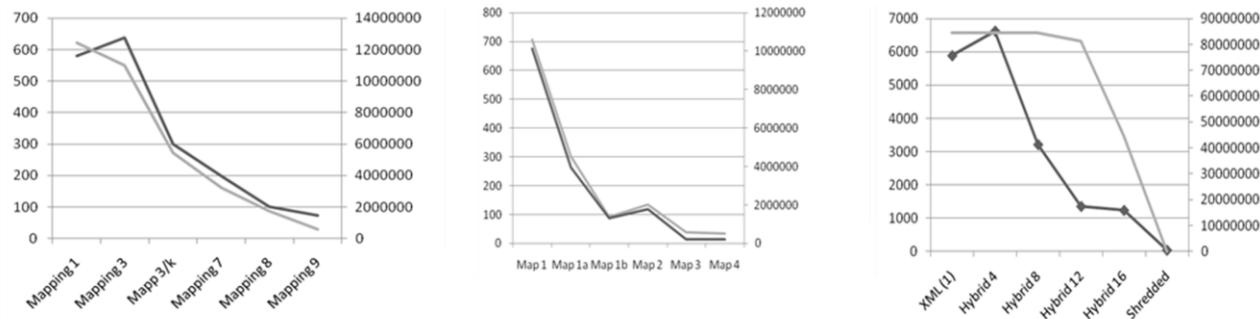


Figure 1. Run times [ms] (black) and data size [bytes] (grey) for PSI-MI (left), UniProt (middle) and Michigan Benchmark (right)

It is important to determine, which properties are relevant for designing efficient hybrid storage. Previous work [12][17] discusses a number of guidelines to take into account during the design process. These guidelines give general advices on how to store data, and we summarize the guidelines from [17] as they provide general goals for this paper.

Guidelines for hybrid XML storage:

1. Keep together what naturally belongs together. Parts of the data that corresponds to a semantic entity is likely to be used together. Therefore it is in many cases a good idea to keep it stored as XML and not shredded into many relations or different representations.
2. Do not shred parts of the XML where the schema allows large variation. As a relational representation is less flexible than XML it is usually preferred to store parts where the schema allows variation as XML.
3. Analyze the data to decide actual variation. The XML schema gives a good intuition of the possible variation of data but it does not give the full picture.
4. Prefer relational representation for elements that are critical for performance. Here, the intuition is to identify the XML elements that are critical for query performance and common queries for the application.
5. Prefer the representation that is required for query results. For the case where the application requires that the result from the query should be returned as XML and not as a relational table shredding is not beneficial.
6. Avoid shredding where new versions of the schema are likely to change.

The above guidelines are easy to use and help the user to design fairly efficient hybrid XML storage for many applications. It should be noted, though, that there are many cases where the different guidelines points in different directions and where the best tradeoff is given by the need of the application. Therefore there is a need for further evaluations and studies.

Exploration and evaluation of alternative solutions is a time consuming task. Methods and tools, to aid the user in

design of hybrid storage, and measurements, that could give hints on how to make choices, are of high importance. Based on the guidelines we can conclude that in order to refine the design guidelines we need to explore properties of the XML structure, the XML schema or DTD and the structure of actual data.

In a preliminary evaluation, we compared the query efficiency with the amount of data stored as XML in the hybrid solution. In our tests, we adopt the shredding principles used in ShreX [14][18] as these principles give a mapping that captures the semantics of a given XML schema for the XML data. To explore hybrid storage we used the extended system HShreX [10][19], which also allows hybrid XML mappings. The general principle behind the mappings of these systems is that complex elements are translated to relational tables. Simple elements and attributes are shredded to a column in their parent table if they occur at maximum once in its parent element. HShreX extends this basic shredding by providing hybrid XML storage, i.e., to allow parts of the structure to be kept as XML in the final database representation. In our study the complexity of the created models varies between one or two relations for the models stored in pure XML to over 100 relations for the fully shredded data models.

The results of these tests are illustrated in Figure 1. The first two graphs show the results for two real data sets from the IntAct [20] and UniProt [21] databases. In this case we can see that the amount of data stored as XML gives a good estimation of the expected query time. For the Michigan Benchmark data [22] the estimation is not as good as for the two other datasets. This means that the amount of data is a good indicator for the performance, but also that further statistics about the data could give us better indicators and aid in effective storage design.

III. AVAILABLE INFORMATION

The general guidelines presented in the previous section show that there are three sources of information that are important to understand storage requirements for a computer application. These are: the general data schema, i.e., the data model (guidelines 1 and 2), samples of data to determine how the data model is used and what parts of the data model are in most common use (guideline 3), and samples of

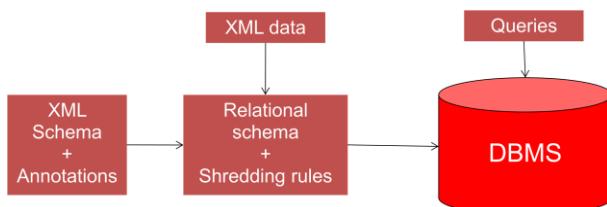


Figure 2. The general architecture of HShreX

queries to determine what kind of queries are often performed for the data (guideline 4 and 5). In this work, we will examine how to use the data model and statistical information for a particular dataset.

As shown in the previous section, the amount of data stored as XML is related to the query performance. However, the prediction we get from simply measuring the amount of data is not enough, we also need to collect more detailed information about the structure of the data. In practice, different parts of the XML schema are populated differently in different data sets. The XML schema carries information about the general structure, but, as for relational databases, the schema does not give a full picture of how this structure is instantiated for a particular dataset. We want to capture this information to create an effective hybrid storage model. In previous work [23], where we worked with generated data, we could see that also the amount of data at various positions in the XML file and the structure of this data had an impact on query performance. We wanted to explore this further and collected the following information:

- Overall statistics for the dataset. With this we mean characterizing the general structure of the dataset. For this purpose we use simple measures, such as, the total number of attributes, elements, and levels in the XML. We also collect the number of elements at each level of the dataset to determine the fan out of the data.
- Diversity of the dataset. To get estimations of diversity we collect the number of elements and attributes for each element or attribute string, at which depths they occur and compare those to the number of overall elements. We also collect information on how many unique search paths occur within the data set and the number of their occurrence.
- Detailed information at each position in the file. This is collected by counting the occurrence of element names at each level in the file. For each combination of parent/child node we count how common the child node is for this parent and collect the minimum, maximum and mean number of times this child occurs for the parent.

Our previous work on generated data has shown that parent/child statistics were of particular interest since this had a large impact on query performance.

IV. A TOOL FOR EVALUATION

To allow easy access to the statistics and aid in evaluating storage alternatives we extended our tool HShreX to include this new information. The new version of the tool can be used to create and evaluate different XML storage models. We start with a description of the general functionality of the system.

The general architecture of HShreX is shown in Figure 2. The system analyses an XML schema and represents it as a tree structure, which facilitates its visual perception. The tree structure helps to easily understand and navigate the schema components as well. The relational schema is likewise created during the schema analyses. Once the database structures are created, large datasets, which corresponds to the currently parsed schema, can be quickly shredded in the database. Each step starting from the XML schema parsing and ending in datasets loading is logged and available for review in a panel under the main work area.

The relational schema is created following the shredding strategy, mentioned above. The actual XML structure is kept by foreign key relations between the created relational tables. These shredding rules are described in [10] and include the following behavior:

- Complex elements are shredded into tables. All tables will get a primary key field named *shrex_id*. If the complex element is not a root element it will also get a foreign key field named *shrex_pid* that points to its parent. This preserves the tree structure in the original XML data. If the complex element can have simple content (i.e., text content), a special field is created in the table to hold any such content.
- Simple elements are shredded into columns in their parent table if they can occur at most once under their parent. If a simple element can occur more than once under its parent it will be outlined to a separate table.
- Attributes are shredded into columns in their parent table.

The user can alter the data shredding rules using HShreX annotations [10]. In this way, the XML data can be represented in purely native, mixed and shredded storage models. The HShreX annotations provide the opportunity to switch rapidly and flexibly between different storage models, create them in a database and evaluate their performance features.

HShreX's user interface provides three panels, which give more details of the schema elements and their mappings. Figure 4 gives an overview of the information on these panels. In the figure we show details for the element *model*. The first panel (top) lists specific details, such as currently applied HShreX annotations, children elements and attributes and their occurrences, for the *model* element in the XML schema tree. In this case the *model* element has three attributes and no annotations have been applied. The second (middle) shows HShreX mapping of the selected element or attribute in the tree. Following our translation rules, *model* is translated into a relational table, with its three XML

The figure consists of three vertically stacked screenshots of the HShreX application interface.

- Top Panel (XML Schema):** Shows the "Schema Tree" pane with nodes like sbml, notes, annotation, level, version, and metad. The "Annotations" tab is selected, displaying a table for the "Complex Element /sbml/model". The table has columns "Property" and "Value", showing "Name" as "model", "Path" as "/sbml/model", and "Content Model" as "(notes,annotation),listOfFunctionDefinitions,listOfUnitDefinitions,listOfCompartments". The "Annotations" section is empty.
- Middle Panel (Relational Mapping):** Shows the "Mapping Editor" tab selected. It displays a table for the "MappingInfo MapTo = MAPS_TO_TABLE Table: 'sbml.model'". The table has columns "Field Name", "SQL Type", "SQL Type Length", "isNullable", "isPrimaryKey", and "isForeignKey". Rows include shrex_id (SQL_INT, default value, false, true, false), shrex_pid (SQL_INT, default value, false, false, true), id (SQL_STRING, default value, true, false, false), name (SQL_STRING, default value, true, false, false), and metad (SQL_STRING, default value, true, false, false).
- Bottom Panel (Relational Schema):** Shows the "Relational Schema" tab selected. It displays a table for the "sbml.model" table. The table has columns "Field Name", "SQL Type", "SQL Type Length", "isNullable", "isPrimaryKey", and "isForeignKey". Rows include shrex_id (SQL_INT, default value, false, true, false), shrex_pid (SQL_INT, default value, false, false, true), id (SQL_STRING, default value, true, false, false), name (SQL_STRING, default value, true, false, false), and metad (SQL_STRING, default value, true, false, false).

Figure 4. HshreX main panels, XML schema (top), relational mapping (middle) and relational schema (bottom)

attributes translated to attributes in the relational table. Note, in particular, the attributes *shrex_id* and *shrex_pid* used to keep the relational structure. The full relational schema and their relations are available in the third panel (bottom of Figure 4).

In this work, the user interface was extended in two directions – to provide more convenient work with HShreX annotations and to visualize more information for a particular dataset.

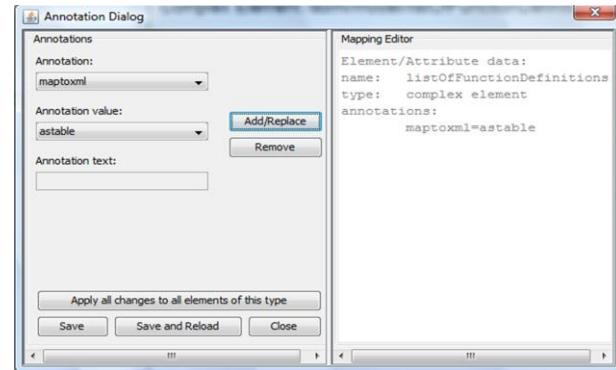


Figure 3. Add/remove annotations dialogue

A. Annotating the data

Important for allowing fast evaluations is easy change of the shredded representation of the data. Therefore, HShreX allows the default shredding rules to be influenced via annotations. The supported annotations were originally developed for relational shredding of XML [14] and extended to allow hybrid XML representation [19]. To get a better understanding of the functionality we give an overview of the most important annotations:

maptoxml – makes this part of the XML tree to be stored natively. The annotation can be used on both complex and simple elements.

ignore – this part of the XML tree will be ignored, i.e., it will not be represented in the resulting data model.

outline – used on simple elements (or attributes) where it is desired that they should be stored in a separate table.

withparenttable – used to merge a child with its parent in order to reduce the number of tables in the model. This annotation can be used only for children with a single occurrence in the parent.

tablename – can be used to simply rename a table but a more powerful use is to merge two tables that do not have a parent/child relationship (in those cases the annotation described above, *withparenttable*, is used).

These annotations allow a rapid change of shredded hybrid storage model. However, in the original system the user had to open and edit the XML schema textually. This was rather complicated and slowed down the process. Therefore we extended the system with a dialog, allowing the user to alter annotations directly from the schema tree, which is shown in the left pane of HShreX.

Figure 3 shows the dialog that facilitates manipulation of HShreX annotations. While navigating in the schema tree, we can open the dialog for the element or the attribute of interest and process its annotations. The dialog provides functionality for adding annotations, updating, i.e., changing values of available annotations and deleting annotations. Since some combinations of annotations for an element or an attribute are not valid, we validate each annotation regarding

the already available annotations prior to adding. A useful feature is provided through the “Apply all changes to all elements of this type” button, i.e., the currently added/removed annotations will be applied to all elements of this type in the XML schema with a single action. The basic data and the annotations, which apply to the element or the attribute of interest, are listed in the right side of the dialog.

B. Statistical analysis of the dataset.

The second improvement in the user interface is orientated towards the statistical information available for a particular dataset. HShreX obtains this information by analyzing a set of sample XML files representing the dataset. We collect the information described in Section III above. However, for designing the interface it was important to make the statistics easy available for the user at the time when it was needed. Therefore we wanted to integrate statistical information into the HShreX user interface as far as possible.

In HShreX detailed information, for the element or the attribute of interest and its children elements and attributes, is presented in the schema tree when a particular dataset is loaded to the database in use. The resulting interface is shown in Figure 6. When data is loaded into the tool, statistical information is shown in the XML Schema Tree (left part of the figure) and additional information is presented in the XML Schema pane (right part of the figure). To start with the statistics show how common the selected element is (in this case the element *model*). The first three lines in the pane show that there are 251 occurrences of the elements *model*, all of them on the second level in the XML file and in this particular position (path) of the files. As a contrast the same value for the element *speciesReference* is shown in Figure 5. From this statistics we can derive that this element is very common, all occurrences are on level 6 in

Property	Value
Name	speciesReference (15875)
Path	/sbml/model/listOfReactions/reaction/listOfReactants/speciesReference (8145)
Content Model	((notes,annotation),(stoichiometryMath)), group type = sequence group
Attributes	stoichiometry (234) species (18733) metaid (24453)
annotation	
speciesReference	6374/6374 (1, 1.2778475054910574, 4)
metaid	

Figure 5. Statistics for the *speciesReference* element.

the file, but only a bit more than half of them in this particular path.

The remainder of the figures in the XML Schema panel shows for each occurrence of child element or attribute occurring in the selected element the total number of occurrences on the document.

The XML Schema Tree (left in Figure 6) gives more information on the structure of the data. For each child element of *model* it shows how common these are as children to *model*. For instance, *listOfCompartments* occurs in all occurrences of *model* while *listOfRules* only occurs in 129 occurrences of *model*. This information is of particular interest when designing the hybrid model as common elements are often beneficial to shred into relations. Three different colors are used to facilitate user's perception and to show how many times a particular child node appears under its parent element, i.e., different children nodes are colored depending on their frequency of appearance. Thus, the user gets fast and highly useful overview of child nodes and can prioritize his next studies based on this information.

In addition, the figures within parenthesis show how the minimal, mean and maximum number of occurrences for each child elements occurs for this element. For *model*, we can see that each of the child elements occurs exactly once (when they are available). For our second example the

The screenshot shows the HShreX interface. On the left is the 'Schema Tree' pane, which displays the XML structure of the SBML file. Nodes are color-coded: yellow for common children, orange for less common, and green for very common. The 'model' node is highlighted in yellow. Below it, 'listOfFunctionDefinitions', 'listOfUnitDefinitions', 'listOfCompartments', 'listOfSpecies', 'listOfParameters', 'listOfRules', 'listOfReactions', and 'listOfEvents' are also visible. On the right is the 'Complex Element /sbml/model (251)' statistics pane. It shows the following details:

Property	Value
Name	model (251)
Path	/sbml/model (251)
Content Model	((notes,annotation),(listOfFunctionDefinitions,listOfUnitDefinitions,listOfCompartments,listOfSpecies,listOfParameters,listOfRules,listOfReactions,listOfEvents))
Attributes	id (26778) name (14710) metaid (24523)
Child Elements of Simple Type	notes (251) annotation (251)
Child Elements of Complex Type	listOfFunctionDefinitions (21) listOfUnitDefinitions (194) listOfCompartments (251) listOfSpecies (237) listOfParameters (197) listOfRules (129) listOfReactions (235) listOfEvents (55)
Annotations	

At the bottom of the interface, a status bar indicates: "Parse of XML file 'BIOMD0000000250.xml' completed. We're not validating against any schemas listed in the XML file. Starting parse of XML file 'BIOMD0000000251.xml'".

Figure 6. Statistics for the Biomodels dataset as used in the HShreX tool.

	Reactome	BioModels
Files	1	251
Levels	6	8
Total elements	31502	93673
Total Attributes	38062	124756
Elements on each level		
	1	250
	1	254
	3	1814
	9144	20008
	8358	32375
	13995	31020
		7951
		1
Total depth	6	8
Mean depth	5,15	5,2
Unique elements	13	35
Unique paths	14	70

Table 1: Statistical overview of two our selected datasets

statistics in Figure 5 shows that each occurrence of *listOfReactants* has one to four child elements (*speciesReference*), the mean number of occurrences is 1.3. The amount of statistical data visualized in the schema tree is small, however, our experience have shown that it is the most useful part of the information available for the dataset. The schema tree representation of statistical information aids the user decision on what annotations are appropriate to be used for a particular dataset and helps to construct proper queries with higher efficiency. Further, the statistics can help the user to create indexes and optimize queries. The other part of the statistical data described in the previous section can be found in “Open Main Statistics” and “Open All Statistics” dialogs under the “File” menu. In addition HShreX can give a summary of all statistics. This summary

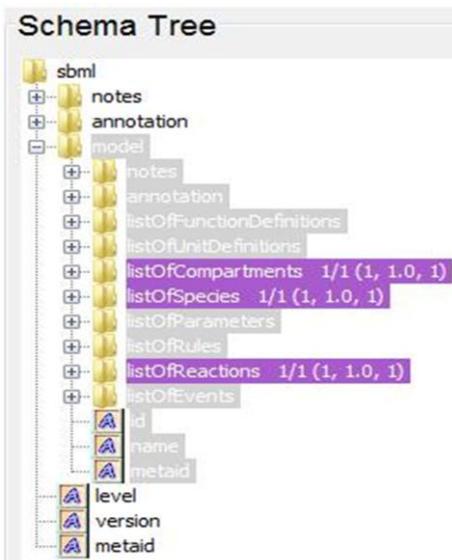


Figure 8. Analysis for the Homo Sapiens dataset.

Figure 7. Statistical data for the *reaction* element. Reactome (top) and BioModels (bottom).

also contains some general facts about the file collection, such as, total number of elements attributes and characters, the maximum and mean depth of the XML data, the total number of unique elements and paths in the data. This data gives a very quick overview of the dataset before designing the storage solution.

V. STATISTICAL ANALYSIS OF DATA

In this section, we show how the statistics can be used to explore two selected datasets. For the study we have selected two datasets represented in the SBML 2.1 (Systems Biology markup Language version 2.1) XML schema [24] XML standard. To explore the benefit of our tool and the statistical information, we used it for designing hybrid shredding and evaluate its performance on the Homo Sapiens dataset from the Reactome database [25] and on the BioModels dataset. Reactome dataset contains an export of data from the Reactome dataset and while the BioModels dataset contains simulation models for pathways.

It turns out that the overall structure of the two datasets is very different. A quick overview of the statistical information provided by HShreX is given in Table 1. From the table we can see that the BioModels data is about three times as large as the Reactome data in terms of number of attributes and elements. It is also clear that the Reactome data have less depth and higher fan out than the BioModels data. This means that the data in the first dataset is spread in depth (the data is stored on many levels) and the data in the second dataset is spread in width (the data is populated almost equally within the dataset).

More interesting is, however, that the number of unique elements and especially unique paths is much larger in the BioModels data, this hints that there is much more variety in the BioModels data than in the Reactome data. We can use the user interface to further investigate the differences.

The statistics available directly in the HShreX schema tree for the two datasets are available in Figures 6 and 8. This pane gives detailed information for the occurrence of

the nodes and their parents and presents a clear view of data

```
Shredded:
SELECT a."id", a."name"
FROM sbml_model_listOfReactions_reaction a,
    sbml_model_listOfReactions_reaction_listOfReactants b,
    sbml_model_listOfReactions_reaction_listOfReactants_speciesReference c
WHERE a."shrex_id" = b."shrex_pid"
    AND b."shrex_id" = c."shrex_pid"
    AND c."species" = 'REACT_5251_1_Oxygen';
```

```
Native:
SELECT reaction.query( 'for $i in //reaction/listOfReactants/speciesReference
    where $i/@species = "REACT_5251_1_Oxygen"
        return <Details> {$i/../@id} {$i/../@name} </Details>' ) "data"
FROM sbml_model_listOfReactions_reaction
WHERE reaction.exist("//reaction/listOfReactants/speciesReference
    [@species='REACT_5251_1_Oxygen']) = 1;
```

Listing 1. Sample query for SBML – Query 1

distribution in the particular dataset. For Reactome dataset all data are collected in the *listOfCompartments*, *listOfSpecies* and *listOfReactants* elements. For the BioModels dataset, the data is much more spread over different parts of the XML schema. Figure 7 shows a further analysis of some part of the data, in this case the *reaction*, and shows that the same relation holds, BioModels data is more diversified than Reactome data. This analysis shows us that a hybrid model for Reactome data can be very simplified as only parts of the XML structure needs to be represented. It also shows that for both datasets *reaction* is one element critical for performance and thus important for further studies.

VI. EVALUATING THE APPROACH

Examining the mentioned datasets, using the HShreX interface, we noticed that some of the elements and their parents occur more often than others, thus our research will be more productive if we concentrate on them. In this section, we will discuss how we work with HShreX in two different application domains.

A. Bioinformatics data

Our discussion in the previous section showed that *reaction* and *model* are important elements in our SBML datasets. Therefore in our examples we have applied the HShreX annotation **maptoxml** to the *reaction* and to the

Shredded:

```
SELECT d."species", b."shrex_pid", e."species"
FROM sbml_model_listOfReactions_reaction_listOfReactants b,
    sbml_model_listOfReactions_reaction_listOfProducts c,
    sbml_model_listOfReactions_reaction_listOfReactants_speciesReference d,
    sbml_model_listOfReactions_reaction_listOfProducts_speciesReference e
WHERE c."shrex_pid" = b."shrex_pid"
    AND b."shrex_id" = d."shrex_id"
    AND c."shrex_id" = e."shrex_id"
    AND d."species" = 'REACT_5251_1_Oxygen';
```

Native:

```
Native:
SELECT reaction.query( 'for $react in //reaction,
    $rtant in $react/listOfReactants/speciesReference,
    $prod in $react/listOfProducts/speciesReference
        return <path> {data($rtant/@species)} {data($react/@id)}
            {data($prod/@species)} </path>' ) "test"
FROM sbml_model_listOfReactions_reaction
WHERE reaction.exist("//reaction/listOfReactants/speciesReference
    [@species='REACT_5251_1_Oxygen']) = 1;
```

Listing 2. Sample query for SBML – Query 2

model elements in the XML schema. This particular annotation/value combination has been selected in order to force the HShreX application to store these parts of the data as pure XML in the corresponding database. If we do not apply any HShreX annotations, the data in the datasets is represented in a shredded storage model. HShreX has been forced to represent the data in a hybrid and in a pure native storage models applying the **maptoxml** annotation to the *reaction* and *model* elements respectively.

We have chosen two of the major database servers available on the market and set up their options related to the XML data representation in various configurations. Using the database servers XML storage capabilities we are able to store the XML data with or without associating it with corresponding XML schema. The database servers run on HP Proliant DL380 G6 Server with two Intel Xeon E5530 Quad Core HT Enabled processors running at 2.4 GHz (in total 16 logical processors) and 30 GB RAM.

We have created different SQL queries (exemplified in Listing 1 and Listing 2) and executed them against the three storage models and different database configurations. In Query 1, the simpler among both, we retrieve details for a reaction where one of its participants is specified. In the second query, we join details for reactions and reactions to extract participants and products for all reactions. First we executed the two queries using only the homo sapiens

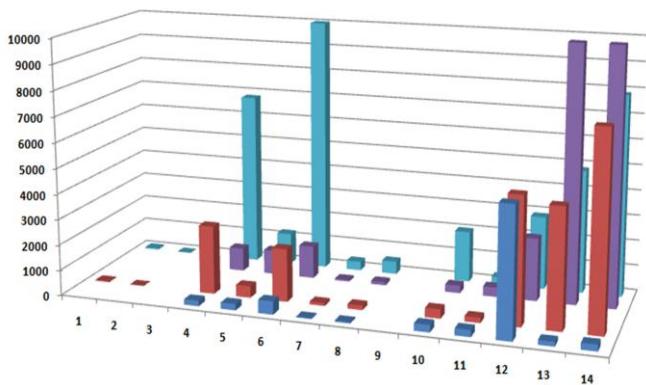
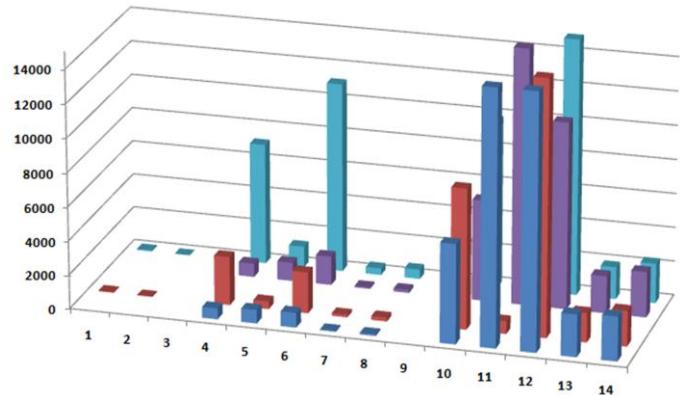


Figure 9. Performance [ms] for Query 1 (left) and Query 2 (right) where: ■ homo sapiens dataset with index, ■ homo sapiens dataset without index, ■ homo sapiens and biomodels datasets with index and ■ homo sapiens and biomodels datasets without index



dataset. After that we loaded both datasets at the same time and evaluated how the response time changes when the size of the data stored in the database increases. The measured performance can be influenced by other processes running on the server. To reduce this influence, the queries from the figures were executed ten times per condition set, and the averages of the results are presented.

First runs were made without any additional optimization. Based on the statistics, proper XML indices, for each variation of database storage options, were created and the same queries were executed again. Thus, we benefit from the statistical information available for a particular dataset in three ways: we can use the statistics to choose the best place for the HShreX annotations regarding our interests and in this way switch flexibly and rapidly between different storage models. We are as well able to create proper, for each storage model, indices based on the view of the data distribution in the particular dataset. A final advantage is that we can optimize our SQL queries not only creating indices but rewriting them based on the data distribution and complexity.

The results from the two different query executions are shown in Figure 9. The equivalent positions on the 'X' coordinate in both of the charts correspond to equivalent condition sets of database storage options. The results from positions 1 and 2 correspond to a fully shredded storage, positions 4 – 8 correspond to a hybrid storage and positions 10 – 14 correspond to a pure native XML storage. Positions 4 – 8 use the same conditions sets of database storage options as positions 10 – 14, however the HShreX annotation is applied to different elements. As we expected, there is a clear relation between the storage model and the query performance, i.e., the execution times are fastest in the shredded storage and slowest in the pure native storage.

Examining the positions 4 – 14 in both result sets we can clearly see that the query performance varies with a different amount for the different database storage options when the size of the data in the database increases. The performance is usually improved when the XML indices are created. It is worth noting that this is not true for position 11 in Query 2 where the performance drops considerably when the index is used. While positions 4 – 8 in the two results sets are comparable, positions 10 – 14 have a lot of differences. Positions 13 and 14 in the first results set have the worst performance among the results for pure native storage while in the second results set they have the best performance. Analyzing positions 13 and 14 in the first result set shows that indices have excellent performance when the size of the data is relatively small and their performance decrease when the data size increases. It is worth noting as well the differences between positions 7, 8 and respectively 13, 14 in the results for Query 1. Positions 7, 13 and 8, 14 respectively have the same database storage options – positions 7 and 8 give the best results while positions 13 and 14 give the worst.

Analyzing the two result sets we can conclude that indices provide better results when used with the hybrid storage than with the pure XML storage. The indices efficiency increases when the size of the data in the hybrid storage increases. During results analysis, we need to

Shredded:

```
SELECT WDDDSO.processor WDP.shrex_pid
FROM workflow_dataflow_processors WDP,
workflow_dataflow_datalinks WDD,
workflow_dataflow_datalinks WDDD,
workflow_dataflow_datalinks_datalink WDDDSI,
workflow_dataflow_datalinks_datalink_source WDDDSO,
workflow_dataflow_processors_processor WDPP
WHERE WDP.shrex_pid = WDD.shrex_pid
AND WDPP.shrex_pid = WDP.shrex_id
AND WDDDS.shrex_pid = WDD.shrex_id
AND WDDDSI.shrex_pid = WDDD.shrex_id
AND WDDDSO.shrex_pid = WDPP.name
AND WDPP.name = module_name;
```

Native:

```
SELECT dataflow.COLUMN_VALUE
FROM workflow_dataflow,
XMLTable('for $i in //dataflow,
          $p in $i/processors/processor/name,
          $d in $i/datalinks/datalink
          where $p = $d/sink/processor
          and $p = $name
          return if(exists($d/source/processor)
                  then <Details>{$d/source/processor}{$i}</Details>
                  else< Details ><processor>null</processor></ Details >''
          PASSING module_name AS "name", "dataflow") dataflow;
```

Listing 3. Input query

consider that the results are also affected from the database servers XML storage capabilities and created indices. The benchmark results are influenced from the data distribution in the datasets as well as the SQL queries construction. The statistical data available in HShreX facilitates and aids our decision where to put HShreX annotations and SQL indices and thus HShreX assists us in fast storage construction.

B. Provenance data

Scientists in the natural sciences use workflow management systems to facilitate their work in development, management and execution of data and computation intensive experiments. These experiments can be described as a sequence of connected activities, where the output of one activity is an input to the following. The experiments are run multiple times with different configurations of parameters where results are produced by each execution. The results obtained from different configurations as well as the configurations itself are highly important for the scientists. They are used for further analysis of the results, as well as sharing and reusing experimental data. The scientific workflow management systems offer tools for describing experiments (workflows), keep track at each step of their evolution and execution and store the resulting data products in an easily reproducible format. Efficient methods for searching and retrieving large amounts of data are essential for the scientists in their everyday work, in this context.

Each workflow system stores the relevant information in its own internal format; however most of them can export the workflows and execution data as XML. Hence usually the workflows are shared in the community in the XML format corresponding to a particular vendor XML schema. Using our tool HShreX and the particular schema the user can obtain a fast overview of the data and to create a storage corresponding to its requirements.

Shredded:

```

UPSTREAM_QUERY (module_name)
SELECT WDDDSO.processor AS preceding_name, WDP.shrex_pid
FROM workflow_dataflow_processors WDP,
    workflow_dataflow_datalinks WDD,
    workflow_dataflow_datalinks_datalink WDDD,
    workflow_dataflow_datalinks_datalink_sink WDDDSI,
    workflow_dataflow_datalinks_datalink_source WDDDSO,
    workflow_dataflow_processors_processor WDPP
WHERE WDP.shrex_pid = WDD.shrex_id
    AND WDPP.shrex_pid = WDP.shrex_id
    AND WDD.shrex_id = WDDD.shrex_id
    AND WDDDSI.shrex_id = WDD.shrex_id
    AND WDDDSO.shrex_id = WDD.shrex_id
    AND WDDDSI.processor = WDPP.name
    AND WDPP.name = module_name;
RETURN UPSTREAM_QUERY (preceding_name)

```

Native:

```

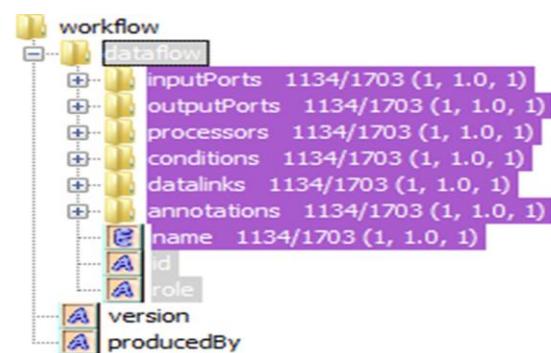
UPSTREAM_QUERY (module_name)
SELECT dataflow.COLUMN_VALUE AS preceding_name
FROM workflow_dataflow,
    XMLTable('for $i in //dataflow,
        $p in $i/processors/processor/name,
        $d in $i/datalinks/datalink
        where $p = $d/sink/processor
        and $p = $name
        return if(exists($d/source/processor))
            then <Details>{$d/source/processor}{$i}</Details>
            else <Details><processor>null</processor></Details>''
    PASSING module_name AS "name", "dataflow") dataflow;
RETURN UPSTREAM_QUERY (preceding_name)

```

Listing 4. Upstream query

There is a set of specific queries that are highly important for scientists in this domain. These are the input and output queries [26] (discover the activities immediately before and after a particular activity), the upstream and downstream queries [26] (discover the activities before and after a particular activity in the whole workflow), activity details query [26] (shows all parameters for an activity), different version queries [27] (show permanent and temporary changes in the workflows structures). Since the input and upstream queries are foundations for more complex queries in this area, they were chosen to show the capabilities and benefits from our tool.

In this experiment, we use a set with approximately 600 files generated by the Taverna [28] workflow management system. Each file contains at least one workflow. So the total dataset contains around 1100 workflows, since an activity can be a workflow on its own. Studying the corresponding schema and the dataset (guideline 3), and taking into account

Figure 10. Statistical data for the *dataflow* element

an additional knowledge for the selected queries (guideline 5), we selected the *dataflow* and the *processors* elements to apply the HShreX annotation **maptoxml**. The structure of the dataset is shown in Figure 10. The *dataflow* element represents the whole experiment (workflow) and the *processors* element represents the activities in it. The *datalinks* element is another important element – it shows how the activities are connected and it has a significant place in the domain specific queries. As described in the previous section, the **maptoxml** annotation will force our tool to store the corresponding parts of the XML as pure XML. When the annotation is applied to the elements the data is represented in pure native and respectively in hybrid storage models.

We have implemented the input and upstream queries (Listing 3 and 4) as SQL functions and executed them against the three storage models. The input query retrieves the activities that immediately precede a given activity. First each workflow is checked for presence of the activity (identified by its name) and when the activity is available the *datalink* elements are explored in order to find the immediately preceding activities. In the upstream query, all activities that precede a given activity in a workflow are retrieved. In order to find all preceding activities in the workflow the input query is executed for every previously selected activity until the beginning of the workflow is reached. Since the upstream query is highly dependent on the structure of the workflow we select and evaluate the query performance for two different activities, which are at

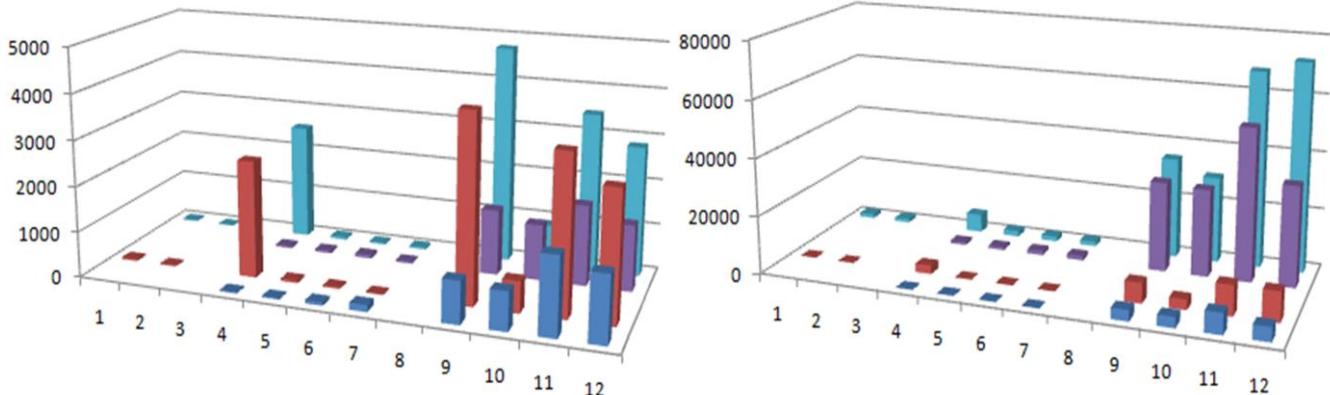


Figure 11. Performance [ms] for the input query (left) and the upstream query (right) where: ■ short path (to the activity regarding the workflow beginning) with index, ■ short path without index, ■ long path (to the activity regarding the workflow beginning) with index and ■ long path without index

different distance from the first activity in the workflow. The two queries were executed on the same database servers and with similar XML storage options as the queries discussed above. Initially, they were executed without any optimizations and then using the statistics in our tool proper indices was created.

The results from the executions of the input and the upstream queries are shown on the left and respectively on the right side in Figure 11. The results on the first two rows (dark blue and red color) on both figures are obtained using an activity close to the beginning of the workflow, while the results on the other two rows are obtained using a distanced (from the beginning) activity. Analogically to the presentation of the bioinformatics data results, the equivalent positions on the 'X' coordinate in both of the charts correspond to equivalent condition sets of database storage options. The results from positions 1 and 2 correspond to a fully shredded storage, positions 4 – 7 correspond to a hybrid storage and positions 9 – 12 correspond to a pure native XML storage. Positions 4 – 7 use the same conditions sets of database storage options as positions 9 – 12, however the HShreX annotation is applied to different elements.

Here, as well as in the bioinformatics data results, the query execution times are fastest in the shredded storage model. The queries performance for the hybrid storage (positions 4 – 7 in both result sets) is very good, sometimes even comparable with the performance in the fully shredded storage. The structure of the queries, where the joins are mainly between shredded relations, has a particular influence on these results. It should be noted that the indices lead to significant improvement in the input query execution time for position 4. Nevertheless, some positions (for instance position 7 on the left figure) in the hybrid storage show a small loss of performance, when the indices are created. A careful examination shows that the query execution times for these positions, obtained in the first run after creating the indices, are very slow. Each query was run ten times per condition set (the average time is shown here) in order to reduce the influence of other processes running on the server at the same time. Although the other execution times for the mentioned positions are very fast, these extreme values influence the average of the results. The query performance, in the pure native XML storage, is usually improved when the indices are created, except the position 10 in both result sets, where the indices lead to worse performance. As expected, due to the upstream query definition, the execution times are dependent on the distance to the selected activity regarding the beginning of the workflow (first two rows on the right figure against the next two rows).

Since each scientific workflows management system has different internal representation of the data, the sharing and reusing of already existing workflows is limited. Thus our current work in the domain of the scientific workflows is orientated towards development of common data representation, optimized for domain specific queries (some of them were mentioned earlier). Since the scientific workflows are best described as directed acyclic graphs, our data model is naturally based on a graph model. Most of the domain specific queries are related to graph traversal and

Query	HShreX storage	Special storage
Input (short path)	43	50
Input (long path)	24	17
Upstream (short path)	183	81
Upstream (long path)	1463	330

Table 2: Comparison between query execution times [ms] in HShreX and in our specially designed storage

other graph operations as well. In this context we have implemented the input and upstream queries in our specially designed model. A comparison between their performance in the graph model and the shredded storage obtained with our tool HShreX is presented in Table 2. Note that although the HShreX shredded storage is not optimized according to the requirements in the scientific workflows domain, it has comparable performance with the storage specially designed for the domain requirements.

VII. RELATED WORK

The work presented in this article combines ideas from several different areas for XML storage. The first is the work on automatic shredding of XML documents into relational databases by capturing the XML structure or based on the DTD or XML schema for the XML data [5][7][14][18]. The intention with these approaches is to create efficient storage for the XML data. The resulting data model is often hard to understand and is usually hidden from the user via an interface providing automatic query translation of XQuery into the model.

The other related area is hybrid XML storage for relational databases. The vendors offer different underlying representation for the XML type, in some cases it is a byte representation of the XML, in other cases it is some kind of shredding of the XML data [8][16][29][30]. In addition, database vendors provide a number of tools to import XML natively or shred the data into the system. These tools are intended for design of one database solution, thus generation and evaluation of alternative solutions become time consuming.

Interesting work [31] has addressed the question of properties of XML data and generating statistical and comparative measurements of XML datasets. However, this work concentrates on overall measures of properties of the dataset and does not consider the more detailed statistical measurements that we have found most useful in our work.

Other related work is found within database optimization [32][33]. Query optimization can rely on statistics of data and query use for fine tuning their performance [9][34]. However, these statistics are often dependent on the internal database representation instead of based on the original dataset as is necessary for our work. It would be interesting to include these measurements in our work to see whether they could give added value to our indicators.

VIII. CONCLUSION AND FUTURE DEVELOPMENT

The extended HShreX tool is very promising and our tests confirms that our tool is very useful for aiding in storage design. Using the tools and statistics improves the evaluation process and makes it possible to compare a high number of alternative hybrid database designs. The statistical analysis gives powerful insight in the structure of data and aids not only in how to shred the data but also in how to construct indices. The added details and experiments, which extend this paper from [1], verify these results.

In particular, we want to compare our set of measurements with the more advanced statistical methods used in [34]. The final goals would be to use the measure to provide suggestions of beneficial hybrid data models for the end user, to further automate the process of storage design. To reach this goal it is crucial to have access to series of data with specific properties to fine tune the indicators and tests. Also for this issue we have made a first solution for generating data with desired properties [23], which can be integrated into our tool.

One bottleneck with our method is that hybrid data models are very complex to query due to the mix of query languages. We are currently using SQL/XML, however, if we consider a user that want to work on the data as if it was XML, this is not feasible. Options are automatic query translations from XQuery to the defined model or to provide a higher level query language for the user.

Another very interesting question is hybrid storage solutions with several DB architectures as a backend, for instance pure native XML databases or specialized databases for graphs or RDF storage. This becomes particularly important for applications where parts of the data contain RDF code or represent graphs as is the case for many system biology standards. We have previously evaluated different combinations [10][13] and would like to include also these options in the HShreX Framework.

ACKNOWLEDGMENT

We acknowledge the financial support from the Center for Industrial Information Technology and the Swedish Research Council. We are also grateful to Juliana Freire for support and fruitful discussions regarding this work and for Mikael Åberg for implementation work on the HShreX tool.

REFERENCES

- [1] L. Strömbäck, V. Ivanova, and D. Hall, Exploring Statistical Information for Applications-Specific Design and Evaluation of Hybrid XML storage., *Proceedings of the International Conference on Advances in Databases, Knowledge, and Data Applications (DBKDA 2011)*, Jan. 2011, pp. 108-113.
- [2] AR. Schmidt, F. Waas, M. Kersten, MJ. Carey, I. Manolescu, and R. Busse, XMark: A Benchmark for XML Data Management, *Proceedings of the International Conference on Very Large Databases (VLDB 2002)*, Aug. 2002, pp. 974–985.
- [3] BB. Yao, MT. Özsü, and N. Khandelwal, XBench Benchmark and Performance Testing of XML DBMSs, *Proceedings of the IEEE International Conference on Data Engineering (ICDE 2004)*, Mar. 2004, pp. 621-633.
- [4] L. Strömbäck, Possibilities and Challenges Using XML Technology for Storage and Integration of Molecular Interactions, *Proceedings of the International Workshop on Database and Expert Systems Applications*, Aug. 2005, pp. 575-579, doi:10.1109/DEXA.2005.154.
- [5] L. Strömbäck, D. Hall, M. Åberg, and S. Schmidt, Efficient XML data management for systems biology: Problems, tools and future vision, *International Journal on Advances in Software*, vol. 2(2-3), 2009, pp. 217-233, Invited contribution.
- [6] D. Florescu and D. Kossmann, Storing and Querying XML Data using an RDMBS, *IEEE Data Engineering Bulletin*, vol. 22(3), 1999, pp. 27-34.
- [7] B. Bohannon, J. Freire, P. Roy, and J. Siméon, From XML Schema to Relations: A Cost-Based Approach to XML Storage, *Proceedings of the IEEE International Conference on Data Engineering (ICDE 2002)*, Feb.-Mar. 2002, pp. 64-75, doi:10.1109/ICDE.2002.994698.
- [8] H. Georgiadis and V. Vassalos, XPath on steroids: Exploiting relational engines for XPath performance, *Proceedings of the ACM SIGMOD International conference on Management of data (SIGMOD 2007)*, Jun. 2007, pp. 317-328, doi:10.1145/1247480.1247517.
- [9] T. Grust, J. Rittinger, and J. Teubner, Why Off-the-Shelf RDBMSs are Better at Xpath Than You Might Expect, *Proceedings of the ACM SIGMOD International conference on Management of data (SIGMOD 2007)*, Jun. 2007, pp. 949-958, doi:10.1145/1247480.1247591.
- [10] L. Strömbäck and D. Hall, An evaluation of the Use of XML for Representation, Querying, and Analysis of Molecular Interactions, In: T. Grust et. al. (Eds) *Current Trends in Database Technology – International Conference on Extending Database Technology 2006 Workshops PhD, DataX, IIDB, IIHA, ICSNW, QLQP, PIM, PaRMA, and Reactivity on the Web*, Mar. 2006, Revised Selected Papers. Lecture Notes in Computer Science, vol. 4254, 2006, pp. 220-233, doi:10.1007/11896548_20.
- [11] I. Mlynkova, Standing on the Shoulders of Ants: Towards More Efficient XML-to-Relational Mapping Strategies, *Proceedings of the International Workshop on Database and Expert Systems Applications*, Sep. 2008, pp. 279-283, doi:10.1109/DEXA.2008.16.
- [12] MM. Moro, L. Lim, and Y-C. Chang, Schema Advisor for Hybrid Relational-XML DBMS, *Proceedings of the ACM SIGMOD International conference on Management of data (SIGMOD 2007)*, Jun. 2007, pp. 959-970, doi:10.1145/1247480-1247592.
- [13] L. Strömbäck and S. Schmidt, An Extension of XQuery for Graph Analysis of Biological Pathways, *Proceedings of the International Conference on Advances in Databases, Knowledge, and Data Applications (DBKDA 2009)*, Mar. 2009, pp. 22-27, doi:10.1109/DBKDA.2009.16.
- [14] S. Amer-Yahia, F. Du, and J. Freire, A Comprehensive Solution to the XML-to-Relational Mapping Problem, *Proceedings of the ACM International Workshop on Web Information and Data Management*, Nov. 2004, pp. 31-38, doi:10.1145/1031453.1031461.
- [15] D. Barbosa, J. Freire, and AO. Mendelzon, Designing Information-Preserving Mapping Schemes for XML, *Proceedings of the International Conference on Very Large Databases (VLDB 2005)*, Aug.-Sep. 2005, pp. 109-120.
- [16] J. Shanmugasundaram, K. Tufte, G. He, C. Zhang, D. DeWitt, and J. Naughton, Relational databases for querying XML documents: Limitations and opportunities, *Proceedings of the International Conference on Very Large Databases (VLDB 1999)*, Sep. 1999, pp. 302-314.
- [17] L. Strömbäck and J. Freire, XML Management for Bioinformatics Applications, *Computing in Science and Engineering*, vol.13, Sep./Oct., 2011, pp. 12-23, <http://doi.ieeecomputersociety.org/10.1109/MCSE.2010.100>.
- [18] F. Du, S. Amer-Yahia, and J. Freire, ShreX: Managing XML Documents in Relational Databases, *Proceedings of the International Conference on Very Large Databases (VLDB 2004)*, Aug.-Sep. 2004, pp. 1297-1300.
- [19] L. Strömbäck, M. Åberg, and D. Hall, HShreX – A Tool for Design and Evaluation of Hybrid XML storage, *Proceedings of the International Workshop on Database and Expert Systems Applications*, Aug. 2005, pp. 575-579, doi:10.1109/DEXA.2005.154.

- International Workshop on Database and Expert Systems Applications*, Aug.-Sep. 2009, pp. 417-421, doi:10.1109/DEXA.2009.33.
- [20] B. Aranda et al., The IntAct molecular interaction database in 2010, *Nucleic Acids Research*, Oct. 2009, pp. 1-7, doi:10.1093/nar/gkp878.
- [21] The UniProt Consortium The Universal Protein Resource (UniProt), *Nucleic Acids Research*, vol. 36(1), 2008, pp. D190-D195, doi:10.1093/nar/gkm895.
- [22] L. Runapongsa, JM. Patel, HV. Jagadish, Y. Chen, and S. Al-Khalifa, The Michigan Benchmark: Towards XML Query Performance Diagnostics, *Information Systems*, vol. 31(2), Apr. 2006, pp. 73-97, doi:10.1016/j.is.2004.09.004.
- [23] D. Hall and L. Strömbäck, Generation of Synthetic XML for Evaluation of Hybrid XML Systems, In: M. Yoshikawa et al. (Eds) *Database Systems for Advanced Applications 15th International Conference, International Workshops: GDM, BenchmarX, MCIS, SNSMW, DIEW, UDM, Apr. 2010, Revised Selected Papers. Lecture Notes in Computer Science*, vol. 6193, 2010, pp. 191-202, doi:10.1007/978-3-642-14589-6_20.
- [24] M. Hucka et al., The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models, *Bioinformatics*, vol. 19(4), 2003, pp. 524-531, doi:10.1093/bioinformatics/btg015.
- [25] Reactome – a curated knowledgebase of biological pathways <http://reactome.org> 25.09.2010.
- [26] L. Moreau et al., Special issue: the first provenance challenge, *Concurrency and Computation: Practice and Experience*, vol. 20(5), 2007, pp. 409–418.
- [27] C. Scheidegger, D. Koop, H. Vo, J. Freire, and C. Silva, Querying and creating visualizations by analogy, *IEEE Transactions on Visualization and Computer Graphics* 13(6), 2007, pp. 1560–1567.
- [28] D. Hull, K. Wolstencroft, R. Stevens, C. Goble, M. R. Pocock, P. Li, and T. Oinn, Taverna: a tool for building and running workflows of services, *Nucleic Acids Research*, 2006, Volume 34, Issue Web Server issue, pp. 729-732.
- [29] K. Beyer, F. Özcan, S. Saiprasad, and B. Van der Linden, DB2/XML:Designing for Evolution, *Proceedings of the ACM SIGMOD International conference on Management of data (SIGMOD 2005)*, Jun. 2005, pp. 948-952, doi:10.1145/1066157.1066299.
- [30] M. Rys, XML and relational Management Systems: Inside Microsoft SQL Server 2005, *Proceedings of the ACM SIGMOD International conference on Management of data (SIGMOD 2005)*, Jun. 2005, pp. 958-962, doi:10.1145/1066157.1066301.
- [31] I. Sanz, M. Mesiti, G. Gurrini, and RB. Llavori, An entropy based characterization of the heterogeneity of XML collections, *Proceedings of the International Workshop on Database and Expert Systems Applications*, Sep. 2008, pp. 238-242, doi:10.1109/DEXA.2008.55.
- [32] G. Gottlob, C. Koch, and R. Pichler, Efficient Algorithms for processing Xpath Queries, *ACM Transactions on Database Systems*, vol. 30, No 2, Jun. 2005, pp. 444-491, doi:10.1145/1071610.1071614.
- [33] J. McHugh and J. Widom, Query optimization for XML, *Proceedings of the International Conference on Very Large Databases (VLDB 1999)*, Sep. 1999, pp. 315-326.
- [34] J. Freire, JR. Haritsa, M. Ramanath, P.Roy, and J. Siméon, StatiX: making XML count, *Proceedings of the ACM SIGMOD International conference on Management of data (SIGMOD 2002)*, Jun. 2002, pp. 181-191, doi:10.1145/564691.564713.

A Proposal of a New Compression Scheme of Medium-Sparse Bitmaps

Andreas Schmidt^{*†}, Daniel Kimmig^{*}, and Mirko Beine[†]

^{*} Institute for Applied Computer Science

Karlsruhe Institute of Technology (KIT)

Karlsruhe, Germany

Email: {andreas.schmidt, daniel.kimmig}@kit.edu

[†] Department of Informatics and Business Information Systems,

Karlsruhe University of Applied Sciences

Karlsruhe, Germany

Email: andreas.schmidt@hs-karlsruhe.de, bem0029@hs-karlsruhe.de

Abstract—In this paper, we present an extension of the WAH algorithm which is currently considered to be one of the fastest and most CPU-efficient bitmap compression algorithms available. The algorithm is based on run-length encoding (RLE) and its encoding/decoding units are chunks of the processor's word size. The fact that the algorithm works on a blocking factor which is a multiple of the CPU word size makes the algorithm extremely fast, but also leads to a bad compression ratio in the case of medium-sparse bitmaps (1% - 10%), which is what we are mainly interested in. A recent extension of the WAH algorithm is the PLWAH algorithm that has a better compression ratio due to piggybacking trailing words, looking “similar” to the previous fill-block. The interesting point here is that the algorithm is also described to be faster than the original WAH version under most circumstances, even though the compression algorithm is more complex. Based on this observation, we extended the concept of the PLWAH algorithm to allow so-called “polluted blocks” to appear not only at the end of a fill-block, but also multiple times inside. This leads to much longer fills and, as a consequence, to a smaller memory footprint, which again is expected to reduce the overall processing time of the algorithm when performing operations on compressed bitmaps.

Keywords – Compressed bitmaps; WAH algorithm; RLE; CPU memory gap

I. INTRODUCTION

One of the main reasons for the work reported here is the increase of the CPU memory gap [1] over the last years. In the context of database applications, processors nowadays are able to process data much faster than the data can be delivered from the main memory to the processor. In many database applications this leads to a situation where the processor(s) spend(s) considerable time waiting for processable data. For this reason, modern processors are equipped with additional cache memory hierarchies which are placed on the processor itself to allow for a much faster access to memory. Accessing a data item that is present in the first-level cache is up to two orders of magnitudes faster than accessing a data item residing in main memory only. Techniques like cache-conscious algorithms [2], [3] or special memory layouts like in *column store* databases allow

for further optimisations to minimise the waste of processor time.

A. Column Stores

In a database system, relations (rows in a database table) are typically stored together physically. This is illustrated in Figure 1 at the bottom left. This storage organisation is called a *row store*. A *column store* database system by contrast stores all values of a specific column sequentially. This storage organisation is visualised on the lower right of Figure 1. The information which column value corresponds to which relation is handled by a tuple identifier (TID). The TID could be stored explicitly with the column values, but is generally given implicitly by the position of the value in the column.

The main advantage of a column store is that only data values from columns that took part in a specific query are loaded into the CPU cache and memory¹, in contrast to a row store, where also unnecessary attributes may be loaded, which are irrelevant in the current context. A disadvantage of column stores, on the other hand, is that in case of updates of existing relations or insertions of new relations, lots of write operations at different positions have to be done, which is more expensive compared to writing data at only one physical location.

Based on the splitting of the relations along their columns, complex predicates have to be processed on a column basis instead of row by row. The reason for this approach is the prefetching behaviour of modern processors. If a dataset is requested by the CPU, not only the requested data item, but also the content of the following memory area is copied into the CPU cache². As a consequence of this behaviour, the decision whether a complex condition is fulfilled by a

¹From secondary storage to main memory and also from main memory to the CPU-cache.

²With every access to the main memory, a full cache line (8-128 Bytes) is loaded. This has the advantage that in case of further requests to the main memory, the requested dataset may already be in the CPU cache.

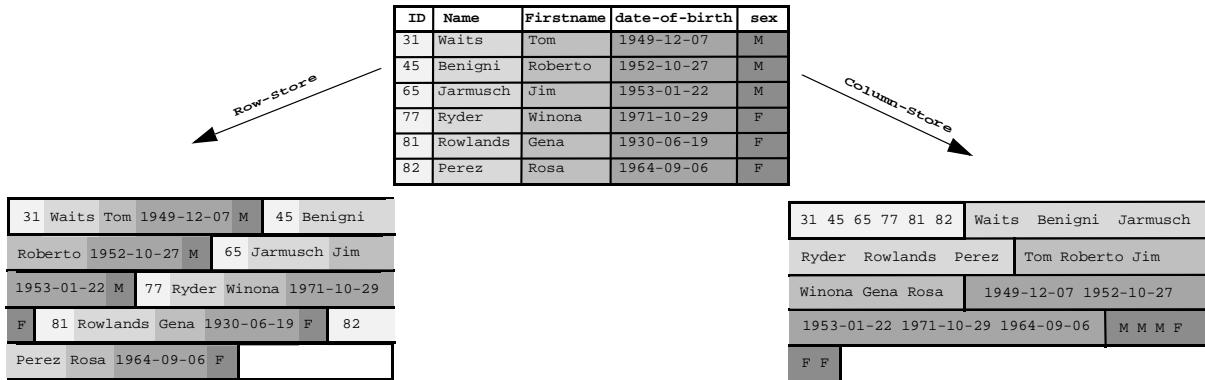


Figure 1. Comparison of memory layouts of a row store and a column store

relation must be delayed up to the examination of the last column.

For this reason, we need an additional datastructure, which remembers the intermediate status of every relation processed so far. This datastructure is called a *positionlist*. A positionlist stores the tuple-ids (TIDs) of the currently qualified relations. The processing of a complex condition generates a positionlist for every single predicate which contains the TIDs of the qualified relations. Afterwards, the positionlists must be combined with *and*- or *or*-semantics. Figure 2 illustrates this behaviour for the following query:

```
select id, name
  from person
 where birthdate < '1960-01-01'
   and sex='F'
```

First, the predicates *birthdate < '1960-01-01'* and *sex = 'F'* must be evaluated, which results in the positionlists *PL1* and *PL2*. These two evaluations could also be done in parallel. Next, an *and*-operation must be performed on these two positionlists, resulting in the positionlist *PL3*. As we are interested in the names of the persons that fulfil the query conditions, we have to perform another operation, which finally returns the entries for a column, specified by the positionlist *PL3*. Positionlists store the TIDs in ascending order without duplicates. For this reason, the typical *and/or* operations can be performed very fast. The complexity for both operations is $O(\|Pl_1\| + \|Pl_2\|)$. Furthermore, the two most important operations *and* and *or* can be performed as bitwise logical operations by utilizing the corresponding primitive CPU commands. By exploiting bit-level parallelism, these operations can be performed extremely fast; with every CPU command, 32 or 64 TIDs (depending on the processor architecture) can be processed. If the positionlists are sparse (meaning we have only a small number of bits set), the underlying bitmap could be compressed easily using run-length encoding (RLE) [4] to further reduce the memory that is required to store the data structure. The main

advantage of RLE is that compression and decompression can be carried out very fast compared to other compression methods. Furthermore, by using specialised algorithms, the *and*- and *or*-operations can be performed directly on the compressed lists, which improves the performance even further.

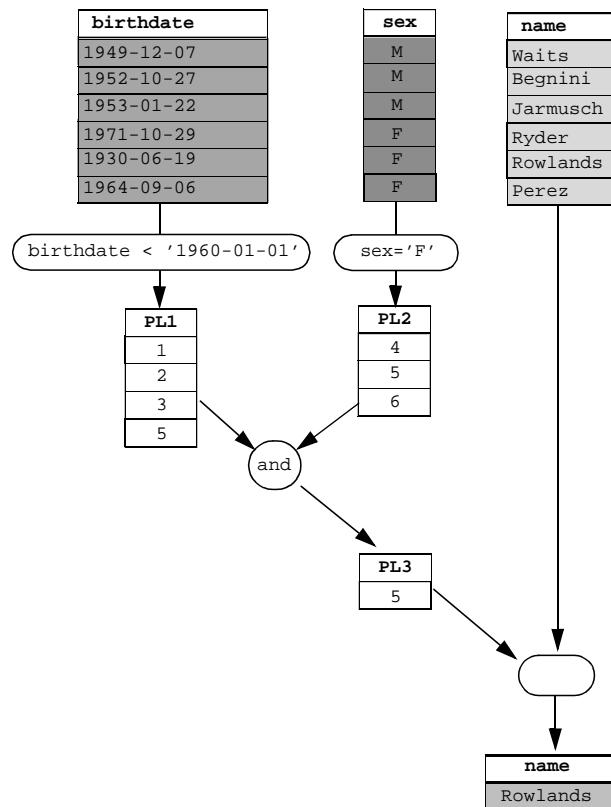


Figure 2. Processing of a query with positionlists

B. Run-Length Encoding

The basic concept of RLE is that for consecutive, identical data elements, only one data element is stored, along with

the number of consecutive occurrences. Such a consecutive sequence of identical data elements is called a *run*, and the number of occurrences is called the *run-length*. RLE is suitable, if many of such runs can be found in a stream of data. Due to these simple basics, compression and decompression can be implemented rather easily compared to more sophisticated techniques like LZ77 [5]. This is an important advantage as because of this simplicity, bitwise logical operations can be carried out without the need to decompress the bitmaps participating in a query.

Figure 3 gives a short example of the application of RLE to a character string.

The rest of the paper is organised as follows. In the next section, we present related work that has been done in the field of bitmap compression. Particularly, we describe the main concepts of the well-known Word-Aligned Hybrid (WAH) [6] algorithm and a recent extension of it, the Position List Word-Aligned Hybrid algorithm [7]. In Section III, we present a new compression scheme that is also based on the WAH algorithm, but introduces a new fill type that is capable of handling “polluted” runs of bits, i.e., runs that contain *mostly* of identical bits. We explain our concept using an example and discuss possible variations of our scheme.

In Section IV, we discuss the results of different experiments performed on synthetic bitmaps to analyse factors that influence the behaviour of our compression scheme.

Our paper will be completed by a short summary and a list of future work that will be tackled once the implementation of our algorithm will be available.

II. RELATED WORK

Apart from their application in the context of positionlists, bitmaps also play an important role in answering multi-dimensional queries in huge datasets. The main idea here is that you have a bitmap for every distinct value of a column [8], [9], [10]. Similar to the example presented earlier, a set bit at position n of a bitmap indicates that a dataset has the specific value at this position. The result set (the TIDs of the relations that match the query criteria) of a multi-dimensional query is then the result of the appropriate *and* or *or* operations on the bitmaps. In this case, the bitmaps represent a special index structure for fast multi-dimensional query processing. This is a very well-known scientific field and a lot of literature as well as implementations of concrete algorithms, i.e., [6] can be found.

The problem in finding a suitable representation form and appropriate algorithms for our positionlist can be mapped onto solutions for multi-dimensional query processing.

The currently fastest algorithm for bitmap operations are based on the *word-aligned hybrid* (WAH) compression algorithm [11]. The main characteristic of this algorithm is

that it is very CPU-efficient, but leads to a bad compression factor in case of selectivities of 1% and above.

As the algorithm already is a few years old and the CPU memory gap is still growing, a difference of up to two orders of magnitude exist in accessing the CPU cache instead of the main memory. Our goal is to develop a new algorithm which does a better job in compressing bitmaps, with selectivities between 0.01% and 10%, but is still IO-bound to further benefit from the increasing CPU memory gap as times goes on.

A. The Word-Aligned Hybrid Algorithm

The WAH algorithm is considered to be one of the fastest bitmap compression algorithms when it comes to the performance of bitwise logical operations on compressed bitmaps. The main reasons for its efficiency are the simplicity of the compression scheme and the word alignment requirement [12]: the size of WAH data units is determined by the word size of the underlying computer architecture, making WAH very CPU-efficient.

The scheme distinguishes between two types of blocks: *literal words* and *fill words*. A literal word stores a heterogeneous sequence of bits (i. e. a sequence that contains a mix of set bits and unset bits). A fill word encodes consecutive, homogeneous sequences of bits (i.e., where all bits have the same value). The most significant bit (MSB) of a word is used to distinguish between a fill word(1) and a literal word(0). In case of a CPU word size of 32, a literal word can thus store 31 bits (hereinafter, we will focus, without loss of generality, on the 32-bit version of the algorithm). Fill words can encode sequences of consecutive either set or unset bits, so one more bit is needed to distinguish a 0-fill word from a 1-fill word (also called the *fill bit*). This leaves 30 bits to encode consecutive homogeneous sequences of bits. This number is called the *fill length*. As WAH imposes the word alignment requirement, the fill length does not describe the total length of such a sequence in bits. Instead, the fill length resembles a multiple of 31-bit-sized, consecutive groups that have the same value. For example, a fill word with a fill length of 2 represents two consecutive blocks, and each block, when decompressed, has a size of 31 bits.

Figure 4 shows the compression of a bitmap of 217 bits (first box). First, the uncompressed bitmap is divided into equidistant parts of 31 bits (second box). Each part is classified as *fill* or *literal* (third box). Finally, consecutive fills with the same bit value are combined to single fills with a corresponding fill length (fourth box).

Figure 5 shows the binary representation of the compressed bitmap from Figure 4. The MSB defines the block type: a 1 (fill) at the beginning of the first, third, and fifth word, and a 0 (literal) for all the other words. In case of a fill, the second bit defines the proper fill type; as we have 0-fills only, all the second MSBs in the fill words are also set to 0. The remaining bits of each fill word are used to encode



Figure 3. Example of run-length-encoded compression

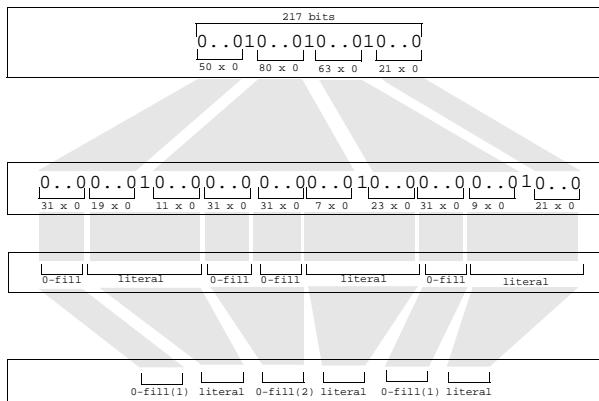


Figure 4. Bitmap compression with WAH

the fill length. Both the header bits and the fill lengths are highlighted in grey.

```
0-fill(1): 10 000000 00000000 00000000 00000000 00000000 1  
literal   : 00 000000 00000000 00001000 00000000  
  
0-fill(2): 10 000000 00000000 00000000 00000000 00000000 10  
literal   : 00 000000 10000000 00000000 00000000  
  
0-fill(1): 10 000000 00000000 00000000 00000000 00000000 1  
literal   : 00 000000 00100000 00000000 00000000
```

Figure 5. Binary representation of the WAH example in Figure 4

The drawback of this algorithm is that in case of medium-spare bitmaps, the *fills* are very short, as every single “pollution”³ causes a running fill to terminate and thus leads to a full literal block. The switch from a fill to a literal (and back to a fill) block is an expensive job in terms of memory.

B. Position-List-Word-Aligned-Hybrid Algorithm

PLWAH is a modified WAH compression scheme that exceeds over WAH in terms of both compression efficiency and the performance of bitwise logical operations. It is based on the observation that in many cases, most of the bits that are available to encode the length of a fill are never needed. PLWAH uses some of these otherwise wasted bits to hold a positionlist⁴. This positionlist is then used to encode a

³pollutionn refers to a single wrong bit

⁴This positionlist must not be mixed up with the positionlists in column stores. Here, the positionlist specifies places of skipped bits.

slightly polluted word that immediately follows a fill word.

So, for example to hold the information about a 32-bit literal word that differs in only one 1 bit from a fill word, 5 bits are needed to describe the position of this wrong bit ($2^5 = 32$)⁵.

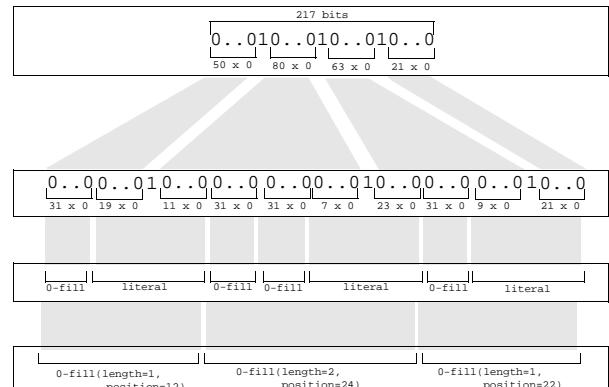


Figure 6. Bitmap compression with PLWAH

Figure 6 shows the compression of the same bit stream used in figure 4 with the PLWAH algorithm. Clearly, all three literals can be piggybacked by the fills that precede them. This leads to an improvement of the compression ratio by about 66% compared to the original WAH algorithm.

With this trick, a reduction by a factor of two can be achieved for certain distributions of data. Otherwise, the maximum length of a fill is reduced by a factor of 2^5 and may reach a maximum of 2^{25} instead of 2^{30} . This could lead to a degradation of up to a factor of 5 for very sparse bitmaps⁶ compared to WAH.

III. CONCEPT

The main difference between WAH/PLWAH and our concept is that we support the concept of *dragged fills*,

⁵for higher pollutions (i.e. 2, 3, ... wrong bits) 10, 15, ... bits are needed to store the position of the wrong bits

⁶densities of $3 * 10^{-8}$ and below

which allows for a small number of false bits inside each word of a fill. The intention here is to obtain longer fills, because the switch from a fill to a literal block and back to a fill is an expensive act in terms of memory.

In contrast to this, our concept does not only allow for one slightly polluted literal at the end of a fill, but it also allows for slightly polluted literals to appear at each position in the fill without reducing the overall length of a fill.

A. Draggled Fill

In addition to the basic WAH word types, our concept requires the introduction of a new block type called draggled fill, which can handle the polluted literals inside a fill. In contrast to the other two block types literal and fill, a draggled fill has a variable length depending on the number of polluted words inside. Hence, three different types of blocks (literal, fill, and draggled fill) must be distinguished. We distinguish a fill from a draggled fill with the third most significant bit, so that a 1-fill is identified by the bit combination of 111, while a draggled-1-fill is identified by 110 (0-fill: 101, draggled-0-fill: 100). The indicator of a literal remains identical to the WAH algorithm (a 0-bit at the most significant bit), which still allows us to store 31 bits in each literal. To decide whether or not a literal can be part of a draggled fill, we first have to define the maximum number of polluting bits that are allowed to occur in such a polluted word.

For a 32-bit version of the WAH algorithm, different degrees of pollution can be defined, from one wrong bit inside 32, 16, and 8 bits (called pollution factor), to 1, 2, or 4 segments containing wrong bits in a complete 32-bit word⁷. Figure 7 presents examples of different pollution factors, each with the maximum number of skipped bits.

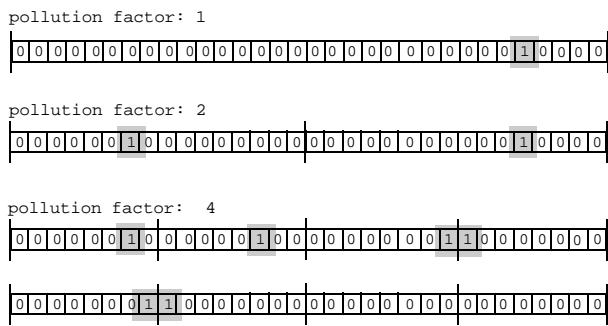


Figure 7. Possible pollution factors for a block

Each polluted 32-bit word needs a fixed number of bits for description. The value of needed bits is dependent on the pollution factor and the maximum length of a fill. In

⁷To be exact, we do not have 32 bits, but only 31 bits as packing unit. But for the sake of straightforwardness the concept is explained based on 32 bit throughout this paper. Keep in mind that, without loss of generality, one bit can be ignored, i.e. the leftmost one.

Table I
MEMORY CONSUMPTION OF DIFFERENT POLLUTION FACTORS

Pollution factor	Memory consumption (in bit)
1	5
2	10
4	16

case of a pollution factor of 1, we only need to specify the position of the wrong bit, which could be done with 5 bits ($2^5 = 32$). With a pollution factor of 2, we need 4 bits to specify the position of the wrong bit in the upper half of the word (i.e., the first 16 bits), and another 4 bits to specify the position of the wrong bit in the second half of the word. As there is the possibility that only one of the halves contains a polluting bit, a mask of another 2 bits is needed to specify in which part(s) of the word the pollutions occur. Table I gives an overview of the memory consumption for the remaining pollution factors (pf). The formula for the pollution factor is:

$$mem = \log_2(32/pf) + mem_mask$$

with

$$mem_mask = \begin{cases} 0 & \text{if } pf = 1 \\ pf & \text{if } pf > 1 \end{cases}$$

Additional memory is needed to specify the position of the polluted words. The size is dependent on the maximum length of a fill. If for example the maximum value is 1024 (2^{10}), 10 additional bits are required to specify the position for each polluted 32-bit word in the most simple implementation, where the position is specified by its index inside the run. Later in Section III-C, we will discuss different possibilities to identify the wrong words.

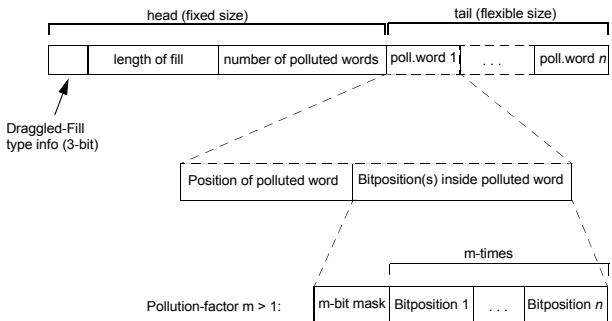


Figure 8. Structure of a *draggled fill* header

Figure 8 shows the structure of a draggled fill. In contrast to literal words and fill words, a draggled fill has a variable size. The fixed-size head contains information about the word type itself, followed by the overall draggled fill length

and the number of polluted words encoded in the draggled fill. The flexible length tail contains information about each polluted word. The concrete size of a polluted word depends on the allowed number of pollutions (as defined by the pollution factor) and the maximum number of polluted words that can be encoded in a single draggled fill. Each polluted word is described by its position inside the draggled fill and the position(s) of the polluting bit(s). In case of a pollution factor > 1 , an additional bitmask is required to determine whether or not a bit is set in the specific part of the word⁸. Having defined the layout of a draggled fill, it is necessary in the next step to determine appropriate values for the maximum length of a draggled fill, the maximum number of polluted words, and the number of allowed pollutions in each polluted word (pollution factor). For this purpose, several experiments were conducted on synthetic bitmaps with varying distributions and densities, as will be reported in Section IV. However, to clarify the ideas behind our concept, we will first demonstrate the function of the algorithm by a simple example. In this example, we assume a pollution factor of 2 and both a maximum fill length and a maximum number of polluted words of 64, meaning that a draggled fill could consist entirely of polluted words.

B. Example

After the introduction of the concept, the effect will now be demonstrated using the example given in Figure 9. In the middle part of the Figure, seven 32-bit blocks can be seen. Except for the fourth and the sixth block, in which the former contains two, while the latter contains one polluted bit(s) (indicated in grey), all remaining blocks contain 0-bits only. The two polluted blocks are shown in detail in the upper and lower part of the figure. The pollution factor is set to 2, meaning that we can accept one wrong bit in every 16-bit of the block at the most. So both polluted words may be incorporated in a draggled fill and the overall length of the fill is 7 words. Besides the overall length, we have to provide additional information for a draggled fill. This information includes:

- The number of polluted blocks
 - The positions of the polluted blocks inside the file
 - Position of the wrong bits inside a polluted block

The maximum number of polluted blocks depends on the maximum length of a *dragged fill* and the number of bits to specify the number. The same holds for the specification of the position of the polluted blocks. In our example, we choose, without loss of generality, a maximum length of a *dragged fill* of 64^9 and a pollution factor of 2. This means that we need 6 bits ($2^6 = 64$) to specify the size of the fill

⁸This extra bit could also be added to the bit position itself, using 0...0 as a marker that no bit skip occurred in this part of the word.

⁹for this simple example a value of 8 would be enough - but this does not seem to be a realistic number

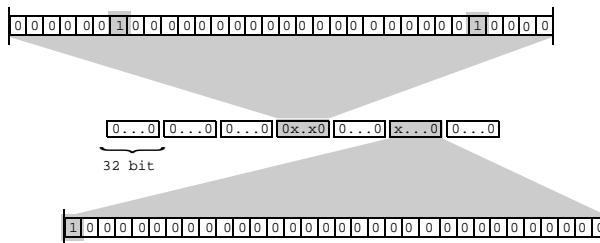


Figure 9. *dragged fill* with two polluted blocks

and another 6 bits to specify the number of polluted blocks inside the fill.

For each polluted block, we also have to provide the information on the position of the block inside the fill and the wrong bits inside. Figure 10 shows a possible memory layout for the above example in the upper part. The first three bits are reserved for the block type, then 6 bits for the fill length field, and another 6 bits for the field indicating the number of polluted blocks.

In the lower 16 bits of the first word, the information about the first polluted word inside the fill is contained. In the defined layout (maximum length: 64, pollution factor: 2), we need exactly 16 bits to specify one polluted word. The first two bits, labelled as “mask”, identify in which of the two halves of the word pollutions occur. Possible values are 01, 10, and 11. The next 6 bits specify the position of the polluted word inside the fill. As a maximum of 1 wrong bit can occur inside one 16-bit block, we need 4 more bits to specify the position (0..15) of the wrong bit inside a single block. As a pollution can occur in the upper 16 bits and/or the lower 16 bits of a word, a total of 8 bits is needed to encode the positions of the polluting bits. In each following 32-bit word, we can now store the information of two more polluted words - one in the upper half, and one in the lower half of the word.

In the lower part of Figure 10, the corresponding bit values for the example in Figure 9 are presented. First, the block type for a *dragged-0-fill* is specified, followed by the information of a fill length of seven with two polluted words. Then, the '11' mask indicates, that there are two skipped bits in the polluted block at position 4 in the fill. The two skipped bits can be found at bit position 9 (first 16-bit word) and bit position 4 (second 16-bit word). In contrast to this, the second polluted block only contains one wrong bit in the first 16-bit word (mask '10'), which can be found at position 15.

The total memory footprint is 64 bits, compared to 160 bits in the original WAH implementation¹⁰ and 128 bits in the PLWAH implementation. Especially in cases of lower selectivity, the proposed concept is superior with regard to

$^{10}160$ bits = 32 bits (0-fill, length: 3) + 32 bits (literal word) + 32 bits (0-fill, length: 1) + 32 bits (literal) + 32 bits (0-fill, length: 1)

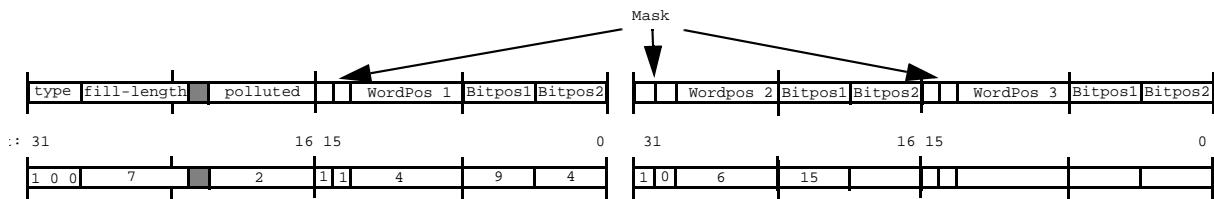


Figure 10. Memory layout of a *dragged-0-fill* with pollution factor 2.

memory footprint. The high memory cost of switching from a fill to a literal block and back can be avoided in many cases. And even in the case where no fills can be found, there is no drawback due to the fact that a literal block can handle 31 bits as in the original WAH-algorithm. However, one small drawback exists in the case of a very high selectivity leading to extremely long fills: because of the new block type, the proposed concept needs one bit more to indicate a fill block, and so a block can contain a maximum of $2^{29} * 31$ bits instead of $2^{30} * 31$.

C. Variants

In the above concept, we divided each 32-bit block into equidistant parts, which can contain 1 wrong bit at the most. This solution was chosen, because it is easy to implement and also CPU-efficient.

Another, more general solution may be not to divide the block into equidistant parts, but to allow a maximum of n -skipped bits to appear inside a 32-bit block. In this case, the memory consumption is a little bit higher, but it is a more general model, which can lead to longer fills.

Instead of specifying the index position of a polluted block, it is also possible to specify the gaps between polluted blocks (incremental encoding [4]). This leads to a smaller memory footprint for each polluted block, because a lower number of bits can be used to specify the increments. In case the next polluted block is too far away to code the distance with the chosen number of bits, the fill has to terminate. Figure 11 gives an example of this encoding. Each gray square represents a 32-bit block (with unique values, polluted and mixed). The full length of the fill is 21 blocks. As you can see, the values of the increments remain small in contrast to the index encoding in the last line, thus allowing for a lower number of bits to encode the fill.

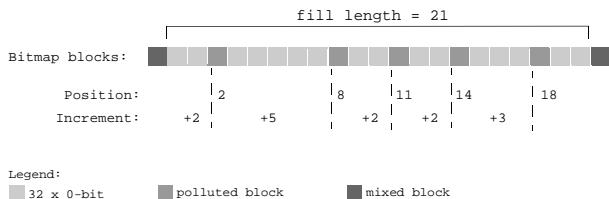


Figure 11. Incremental encoding of “polluted blocks”

All of the above variants require a predefined fixed

number of bits to encode the position of the polluted blocks. Another possible solution would be to use a Rice (Golomb) coding [13]. The idea behind this coding scheme is to use a flexible number of bits to encode arbitrarily long integer numbers. Small, but frequently appearing numbers only need a small number of bits, while unfrequent big numbers need more bits as in a normal coding scheme.

Figure 12 and Figure 13 show a possible encoding of the examples from Figure 4 (WAH) and 6 (PLWAH) for the draggled fill WAH algorithm using incremental encoding of the polluted blocks. As you can see in Figure 13, another reduction of about 33% can be achieved, additionally leaving some bits unused, which would be able to hold another polluted word (if existent). The first three bits indicate the header type (*0-draggled-fill*). The next six bits give the overall length of the fill (7), followed by five bits indicating the number of polluted words inside the fill (3). Depending on the value (n) in this field, $\lceil (n-1)/3 \rceil$ words¹¹ follow to hold the information about each polluted word. The information about the first pollution can be stored in the lower 16 bits of the *dragged-fill-header*. Here in our example, the first polluted word follows after one *0-fill* word and has its pollution at bit 12. The first ten bits in the second word (flexible part) hold the information that the next polluted word follows after a gap of two *0-fill* words and the polluted position is bit 24. With this encoding (5 bits to store the gap), a maximum gap length between two polluted words can be 32. The 'x' characters indicate unused bits.

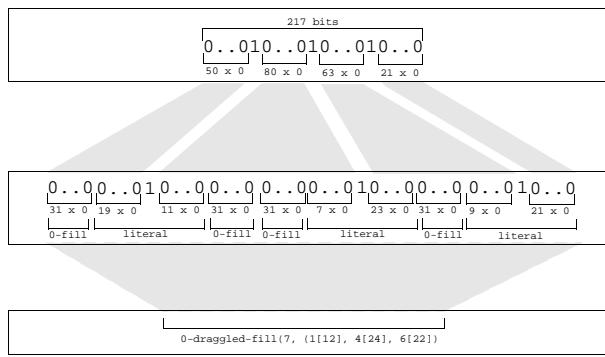


Figure 12. Bitmap compression with DFWAH

$$^{11}\lceil(3-1)/3\rceil = 1$$

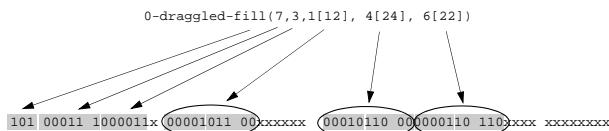


Figure 13. Binary representation of DFWAH from Figure 12

IV. EXPERIMENTS

As the size of a compressed bitmap is one of the most critical factors and we need to define the concrete memory layout for our draggled fill word.

As discussed before, we have some degree of freedom. First of all, we can choose between different pollution factors, we have to define how many bits we allocate to hold the length information or the information for the maximum numbers of pollutions inside a fill which both can restrict the maximum length of a draggled fill. The maximum distance of two pollutions in a fill (see Section III-C) also has a direct impact on the size of a draggled fill.

To obtain a feeling for appropriate settings of these values, we conducted a number of different experiments. In these experiments, we generated long bitmaps with different distributions.

Then, we took these bitmaps and ran our algorithm on it. In contrast to a real implementation, we did not have a maximum size of a fill, a maximum number of pollutions inside a fill or a maximum gap distance between polluted words and so on in this “ideal” implementation. We rather tried to find appropriate values for these parameters.

Input parameters for the bitmap generation are the density and different distributions (see below). The *density* is the fraction of “1” bit inside a bitmap and can be between zero and one. Our experiments focused on densities between 0.005 (0.5%) and 0.1 (10%), representing our medium-sparse bitmaps.

In our experiments we distinguish between different distributions:

- **Uniformly distributed bitmaps:** In a uniform distribution, every possible value (0/1) occurs all the time with the same probability, independently of previous values. In this case, the *density* is the same as the probability that a “1” value occurs. Such bitmaps can be generated easily using the standard random function.
- **Clustered bitmaps:** In a clustered distribution, set bits tend to occur in groups or clusters. Therefore, there is a higher probability, that more consecutive set bits occur than in uniformly distributed bitmaps. Clustered bitmaps typically reflect application data better than uniform distributions [6]. Bitmaps that follow a clustered distribution can be generated easily with a simple two-state *Markov chain*. Figure 14 shows such a finite state machine with probabilities of p and q for changing the state from the prior state. The *density* of such a

distribution can be calculated by $d = p * (p + q)$ [7]. Additionally, the cluster factor f is determined by $f = 1/q$. Typically, the input for the generation of a Markov chain-generated bitmap is the density d and the clustering factor f . In this case, the probabilities p and q are calculated by $q = 1/f$ and $p = q * d/(1 - d)$.

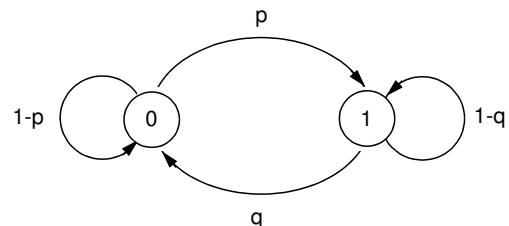


Figure 14. Two-state Markov-chain

A. Length of draggled fills

In a first series of experiments, we examined the potential length of the draggled fills. Figure 15 shows the distribution of draggled fill lengths for different densities between 0.5% and 10% and a pollution factor of 2. As expected, the length of a draggled fill strongly correlates with the density factor. Additionally, an approximate linear correlation between density and average/maximum fill length can be observed. For the interesting densities between 0.005 and 0.1, the average size of a fill is below 20. In Figure 16, the coverage for the different distributions in the previous figure is shown. So, for a density of 0.01, with a maximum fill length of 255 (8 bit memory consumption), 95% of all possible draggled fills discovered in our experiment are covered. Choosing 10 bits for the length field, a coverage of 99.999% of all possible fills can be achieved.

Figure 17 and Figure 18 show the same issue, but with a Markov distribution and a clustering factor of 2. Here, the distribution is more compact compared to the uniform distribution shown before in Figure 15.

Next, we examine a Markov chain-based distribution with higher clustering factors. A higher clustering factor leads to a higher possibility of literal words, but also longer runs of 0 bits appear. Figure 19 shows the distribution for different clustering factors for a distribution with a density 0.01. Obviously, the length increases approximately linearly with the clustering factor.

Resume: The longest runs result from uniformly distributed bitmaps with a low density. For the density range we are interested in, a maximum length between 255 (8 bits) and 1024 (10 bits) for a fill seems to be an appropriate value¹². For the very small number of possible longer runs, a second run has to be started. For densities of about 0.05 (5%) and above, even a maximum length of 32 (5 bits) is ok.

¹²for a pollution factor of 2.

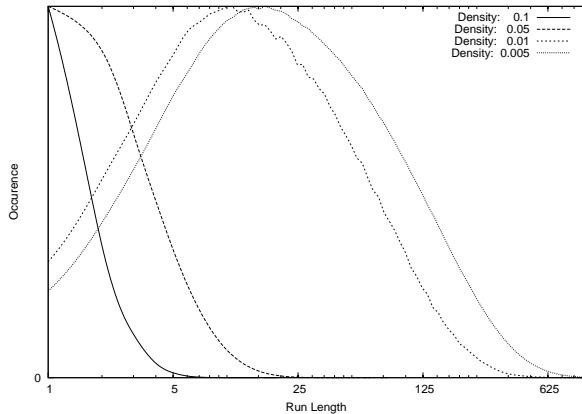


Figure 15. Distribution of dragged fill lengths with uniform distribution (pollution factor 2)

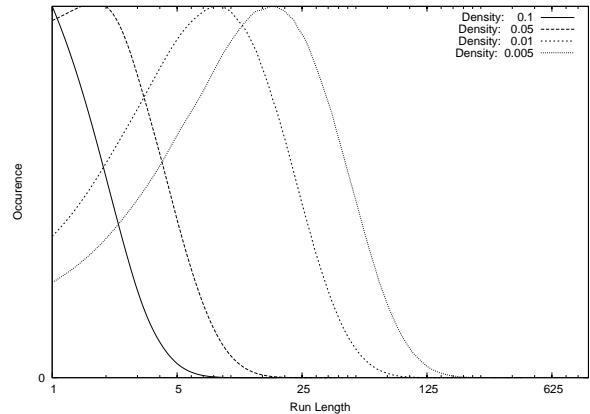


Figure 17. Distribution of dragged fill lengths with a Markov distribution, clustering factor 2 (pollution factor 2)

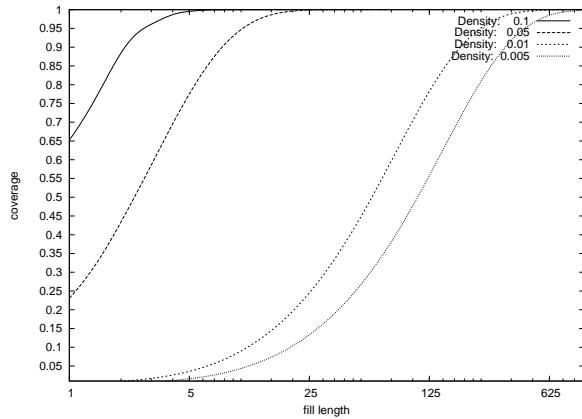


Figure 16. Coverage for different maximum fill lengths from Figure 15

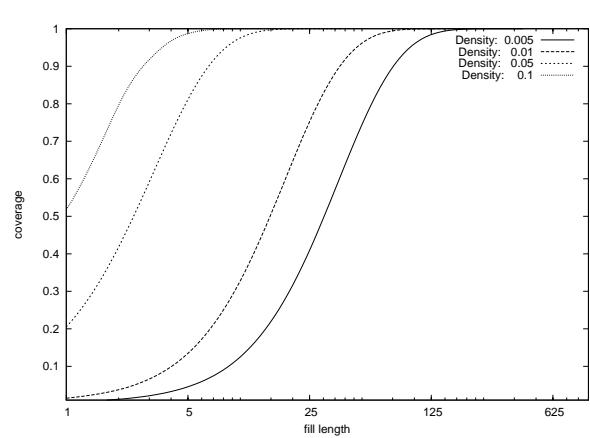


Figure 18. Coverage for different maximum fill lengths from Figure 17

B. Influence of the pollution factor

The next series of experiments are performed to obtain an idea of the influence of the different pollution factors. For a density of 5% (uniform distribution), Figure 20 shows the distribution of the length for different pollution factors. As expected, the run-length increases with the increase of the pollution factor. In all cases, the average number of pollutions inside a fill is about 74%-77%, irrespective of the pollution factor. In the case of 1% density (not shown), the number of pollutions inside a run is between 24% and 26% of the overall fill length. A comparable behaviour can be observed for the other densities of interest.

Figure 21 by contrast, shows the behaviour for the Markov-chain distribution. As in the previous case, the behaviours for different pollution factors are shown. Compared to the uniform distributions, the advantage of using a higher pollution factor is smaller, due to the characteristic of the distribution and the restriction of the positions of wrong bits inside a polluted word (see Section III-A). As a consequence, there is no advantage in using a pollution factor of 4,

compared to a pollution factor of 1 or 2 - even more so as the handling is more CPU-intensive and the memory footprint of a polluted word corresponds to about a factor of 1.6 compared to a pollution factor of 2. The average number of pollutions inside a fill is between 45% and 47%. This lower factor is derived from the fact that in the Markov chain-based distribution used here the '1' bits are more clustered and longer runs of '0' bits appear. An increasing clustering factor for the Markov-chain distribution yields to longer runs compared to lower values (see also the previous experiment in Figure 19), but no further improvements for higher pollution factors.

Resume: Choosing a higher pollution factor yields to longer runs. Because of the restriction in the layout (see Figure 7), a number higher than 4 for the pollution factor does not seem to be meaningful. This is especially true for the more clustered behaviour of Markov chain-generated bitmaps. So the values determined in Section IV-A are still valid for the different pollution factors.

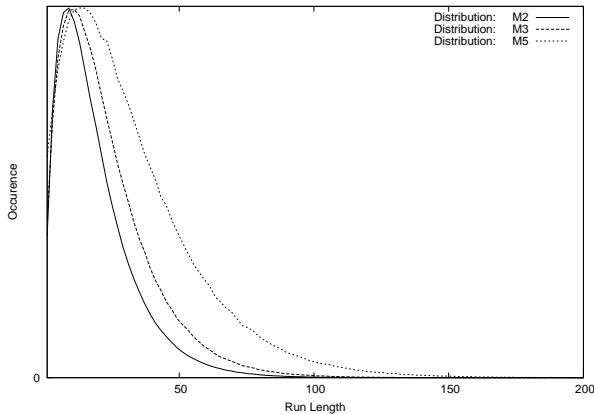


Figure 19. Markov-chain distribution: comparison of the influence of different clustering factors to the run-length (PF: 2, density: 0.01)

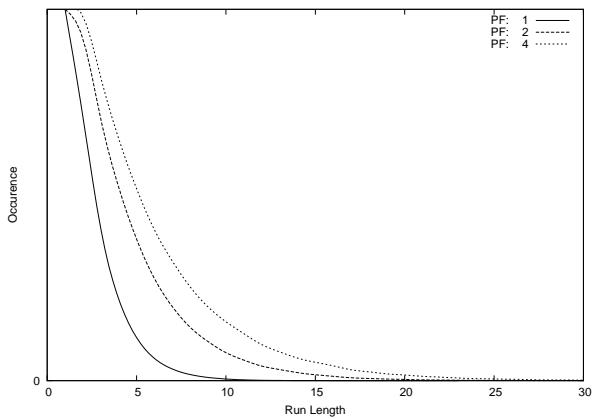


Figure 20. Distribution of run-length for different pollution factors (uniform distribution, density 5%)

C. Distance between pollutions in a draggled fill

As explained in Section III-C, the position of a pollution can not only be expressed by its position index inside the fill, but also by the distance to the previous pollution. This has the advantage, that with a growing size of a fill, the memory consumption to express the next pollution position is smaller.

Figure 22 shows the distribution of the gap width for different densities (uniform distribution, pollution factor 2). For the densities we are interested in, a maximum gap width of 16 (covering 95% of all gaps for a 0.01 density) or possibly 32 (covering 99.99%) is sufficient, which means that we need 5 or 6 bits memory consumption for the length field (the maximum gap width for Markov chain-distributed bitmaps is slightly lower).

Figure 23 shows the coverage for different gap lengths and different densities. So i.e., with a maximum gap length of 32 and a density of 0.005, a coverage of 99% can be achieved. For a density of 1%, this can already achieved

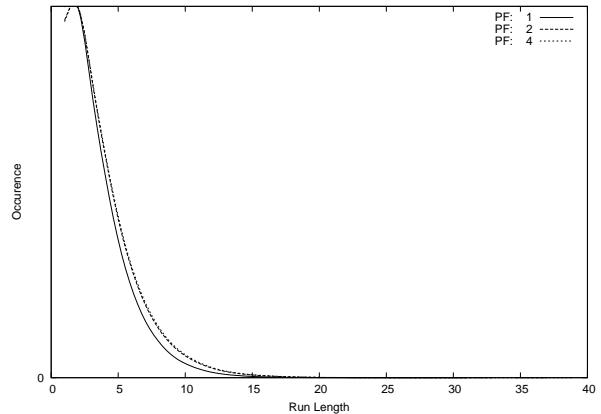


Figure 21. Distribution of run-length for different pollution factors (Markov-chain distribution, clustering factor 2, density 5%, pollution factor 2)

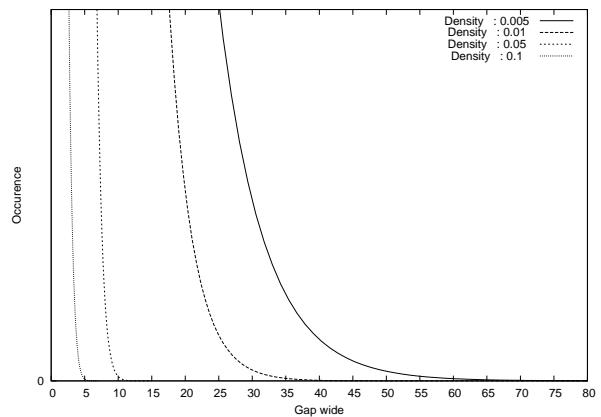


Figure 22. Distribution of gaps between pollutions in a draggled fill (uniform distribution, pollution factor 2)

with a maximum gap length of 16.

More experiments and also some analytical examinations concerning the size of the bitmaps can be found in [14].

V. CONCLUSION

We presented an extension of the WAH algorithm, which is currently considered one of the fastest and most CPU-efficient compression techniques for bitmaps. However, in the case of a selectivity of 1% and more, the compression behaviour of WAH is unsatisfying. The reason for this behaviour is the blocking factor of 32, which requires packing of a minimum of 31 bits. Thus, even a single skipped bit leads to a full literal block, which holds 31 uncompressed bits.

Our contribution handles this problem by allowing so-called polluted words to be part of a fill. A polluted word is a block which has a limited number of wrong bits. The idea is to describe the position of the polluted words in the fill and the wrong bits inside it instead of storing the whole

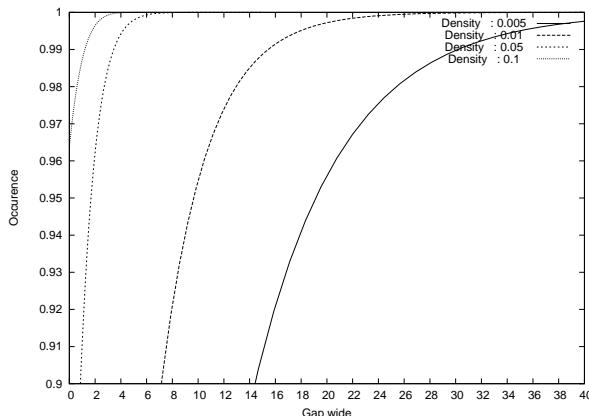


Figure 23. Coverage of gap width from the distribution in Figure 22

literal word, which takes less memory than ending a fill, starting a new literal block, and after that starting a new fill.

After presenting the concept of our improved algorithm, we ran a number of different experiments, to obtain a feeling of the behaviour of our algorithm for different distributions (uniform, Markov with different clustering factors), different pollution factors, and different implementation variants.

VI. FUTURE WORK

Currently, final implementation of our concept is not yet finished, but we already have a functional prototype without any optimisations. Our next step will be to integrate the algorithm in the WAH codebase. To do so, we will completely rewrite of our functional prototype. Once we have our implementation finished, we plan a number of tests with different values for selectivity, reflecting both synthetical and real world data in order to compare both the compression ratio and the execution time of the different operations. Depending on the results, we will eventually implement different variants of our algorithm, which we discussed in Section III-C.

REFERENCES

- [1] A. Schmidt and M. Beine, "A Concept for a Compression Scheme of Medium-Sparse Bitmaps," in DBKDA 2011, Proceedings of the Third International Conference on Advances in Databases, Knowledge, and Data Applications, St. Maarten, The Netherlands Antilles, 2011, pp. 192–195.
- [2] S. Manegold, P. A. Boncz, and M. L. Kersten, "Optimizing database architecture for the new bottleneck: memory access," *The VLDB Journal*, vol. 9, no. 3, 2000, pp. 231–246.
- [3] T. M. Chilimbi, B. Davidson, and J. R. Larus, "Cache-conscious structure definition," in PLDI '99: Proceedings of the ACM SIGPLAN 1999 conference on Programming language design and implementation. New York, NY, USA: ACM, 1999, pp. 13–24.
- [4] I. H. Witten, A. Moffat, and T. C. Bell, *Managing gigabytes* (2nd ed.): compressing and indexing documents and images. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999.
- [5] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE TRANSACTIONS ON INFORMATION THEORY*, vol. 23, no. 3, 1977, pp. 337–343.
- [6] K. Wu, E. J. Otoo, and A. Shoshani, "Optimizing bitmap indices with efficient compression," *ACM Trans. Database Syst.*, vol. 31, no. 1, 2006, pp. 1–38.
- [7] F. Deliège and T. B. Pedersen, "Position list word aligned hybrid: optimizing space and performance for compressed bitmaps," in EDBT '10: Proceedings of the 13th International Conference on Extending Database Technology. New York, NY, USA: ACM, 2010, pp. 228–239.
- [8] P. O'Neil and E. O'Neil, *Database: Principles, Programming, Performance*. San Francisco, CA: Morgan Kaufmann, 2001.
- [9] J. W. Hector Garcia-Molina, Jeffrey D. Ullman, *Database System Implementation*. Prentice-Hall, 2000.
- [10] J. Wu, "Annotated references on bitmap index." [Online]. Available: <http://www-users.cs.umn.edu/~kewu/annotated.html>, retrieved: december, 2011.
- [11] K. Wu, E. J. Otoo, and A. Shoshani, "Compressing bitmap indexes for faster search operations," in SSDBM '02: Proceedings of the 14th International Conference on Scientific and Statistical Database Management. Washington, DC, USA: IEEE Computer Society, 2002, pp. 99–108.
- [12] K. Wu, E. J. Otoo, and A. Shoshani, "A performance comparison of bitmap indexes," in CIKM '01: Proceedings of the tenth international conference on Information and knowledge management, ser. New York, NY, USA: ACM, 2001, pp. 559–561.
- [13] S. W. Golomb, "Run-length encodings," *IEEE-IT*, vol. IT-12, 1966, pp. 399–401.
- [14] M. Beine, "Implementation and Evaluation of an Efficient Compression Method for Medium-Sparse Bitmap Indexes," Bachelor Thesis, Department of Informatics and Business Information Systems, Karlsruhe University of Applied Sciences, Karlsruhe, Germany, 2011.

Turning Large Software Component Repositories into Small Index Files

Marcos Paulo Paixão, Leila Silva

Computing Department
Federal University of Sergipe
Aracaju, Brazil

marcospsp@dcomp.ufs.br, leila@ufs.br

Talles Brito, Gledson Elias

Informatics Department
Federal University of Paraíba
João Pessoa, Brazil

talles@compose.ufpb.br, gleudson@di.ufpb.br

Abstract—Software component repositories have adopted semi-structured data models for representing syntactic and semantic features of handled assets. Such models imply key challenges to search engines, which are related to the design of indexing techniques that ought to be efficient in terms of storage space requirements. In such a context, by applying clustering techniques before indexing component repositories, this paper proposes an approach for reducing the number of assets in the repository, and consequently, the size of index files. Based on an illustrative repository, outcomes indicate a significant optimization in the number of assets to be indexed, and, as a consequence, produces significant gains in storage requirements. Besides, it has been assessed in terms of two different clustering evaluation methods, evincing that the proposed approach can be considered a good clustering algorithm because produces compact and well-separated clusters.

Keywords - Component repositories; clustering techniques; indexing.

I. INTRODUCTION

By enabling different software developers to share software assets, software component repositories have the potential to improve software reuse level. However, reuse of software assets is in general a hard task, particularly when search and selection must be conducted over large-scale asset collections. Therefore, in repository systems, it is important the development of search engines that can help searching, selecting and retrieving required software assets.

According to Orso *et al.* [1], the aim of a repository system is not to store software assets only, but also metadata describing them. Such metadata provides information employed by search engines for indexing stored assets. In such a direction, as endorsed by Vitharana [2], component description models can adopt high level concepts for describing component metadata, making possible to express syntactic and semantic features, and so, facilitating developers to search, select and retrieve assets. In practice, currently available component description models have adopted approaches based on semi-structured data, more specifically XML, allowing structural relationships among elements to aggregate semantic to textual values. As examples, it can be mentioned RAS [3] and X-ARM [4].

However, indexing techniques based on textual restrictions are not efficient for semi-structured data. Such

techniques are unable of indexing structural relationships among terms, compromising query precision with false-positives. Thus, the adoption of semi-structured data implies challenges related to the design of indexing techniques that ought to be efficient in terms of storage space requirements, processing time and precision level of queries, which can be constrained by textual and structural restrictions.

Several proposals can be found in the literature for dealing with such problems. Despite their relevant contributions, existing techniques do not meet storage space and query processing time requirements [5], and also query precision level [6]. In such a scenario, the proposal presented by Brito *et al.* [7] represents a noticeable indexing technique based on semi-structured data, which can be considered precise and efficient in terms of query processing time, but suffers from problems related to storage space requirements. Such problems occur because generated index files are bigger than the input database. Thus, in the context of large-scale software component repositories, it is still a challenging open issue to design indexing techniques that minimize the storage space requirements without excessively impacting on query processing time and precision.

In such a context, based on the adoption of clustering techniques, this paper proposes an approach for reducing the number of assets in the repository, and consequently, optimizing the storage space requirements. It is an extended and improved version of [8]. The clustering heuristic proposed is based on the classical hierarchical algorithm and K-means [9]. Taking into account a large-scale component repository, the proposed approach identifies clusters (groups) of similar software assets and generates new representative assets, which in turn must be handled by the indexing technique supported by the search engine of the repository. Each representative asset has a simplified description, also based on semi-structured data, which makes reference to all original assets that belong to its cluster of similar assets. In order to do that, the paper also proposes a similarity metric that has the aim of indicating the set of assets that belongs to the same cluster. The bigger the similarity among assets in the repository, the lesser is the number of identified clusters, and as a result, the lesser is the number of representative assets that must be indexed by the search engine, enabling to save storage space. In order to validate the proposed approach, a random database composed of 14.000 assets has

been generated and results indicate that there is a significant optimization in terms of the number of assets to be indexed.

The remainder of this paper is structured as follows. Section II describes related techniques, evincing the original initiative of applying clustering techniques in the context of indexing software component repositories. The adopted component description model, called X-ARM, is briefly presented in Section III, identifying the main types of assets and their relationships. Then, Section IV presents the proposed clustering approach for reducing the number of assets to be indexed, and so, optimizing storage space requirements. After that, some outcomes observed in a preliminary evaluation performance are presented in Section V. In conclusion, Section VI presents some final remarks and delineates future work.

II. RELATED TECHNIQUES

Taking into account that the problem of data clustering is NP-hard, several heuristics have already been proposed. Xu and Wunsch [10] present an interesting review of the research field. In [11], Feng shows that clustering algorithms, in particular, hierarchical algorithms and *K*-means [9], are equivalent to optimization algorithms of a fitness function.

Clustering techniques have been used in several software engineering domains. For example, Mancoridis *et al.* [12] applied clustering in the domain of software maintenance, by introducing the concept of software modularization as a clustering problem for which search is applicable. A tool called Bunch [13] is proposed allowing the application of several clustering heuristics to perform search based software modularization. Chiricota *et al.* [14] investigates the application of clustering techniques in the domain of reverse engineering, in particular, adopting such techniques to recover the structure of software systems. Wu *et al.* [15] compares several clustering approaches proposed in the context of software evolution. In [16], Li *et al.* proposes the adoption of clustering techniques for encapsulating software requirements. Cohen *et al.* [17] showed how search based clustering algorithms could be applied to improve garbage collection in Java programs.

Although clustering techniques are applied in several problems of software engineering, for the best knowledge of the authors, these techniques have never been adopted in the context of indexing software components repositories. Therefore, it seems an original contribution to apply such techniques when indexing component repositories.

III. THE X-ARM MODEL

In order to express syntactic and semantic features of software components, Frakes [18] suggests the adoption of component description models, which provide a set of information that allows search systems to index and classify all types of related assets. In such a direction, this paper explores the X-ARM description model, which adopts a XML-based semi-structured data model, expressing not only

syntactic information but also semantic properties [4]. Besides, X-ARM enables describing several types of software assets, which can be produced in component-based development processes, proving the required semantic for representing their relationships.

As illustrated in Fig. 1, X-ARM allows describing component and interface specifications, as well as component implementations. The component and interface specifications can be described in a way that is independent or dependent of component model. On the one hand, independent specifications do not take into account any feature or property of component models, such as CCM, JavaBeans, EJB and Web Services. On the other hand, dependent specifications ought to consider features and properties related to the adopted component models.

In X-ARM, both dependent and independent interface specifications are described as a set of operations. Each operation has a name, a set of input or output parameters and a return value. In component-based development processes, dependent interface specifications must be in conformance with their independent counterparts. So, in Fig. 1, it can be observed that dependent interface specifications must reference to their respective independent interface specifications.

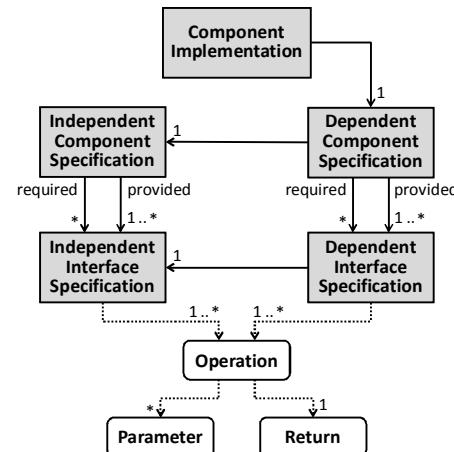


Figure 1. Relationships between artifacts.

Dependent and independent component specifications can make reference to a set of provided and required interface specifications. However, it must be noticed that independent component specifications can refer to independent interface specifications only. Similarly, dependent component specifications can refer to dependent interface specifications only. In component-based development processes, dependent component specifications must be in conformance with their respective independent counterparts. Therefore, note that dependent component specifications must make reference to their respective independent component specifications.

In summary, dependent interface and component specifications must be in conformance with their respective independent specifications. Besides, for each independent

specification, several dependent specifications can be described, each one in conformance with a given software component model.

In a similar way, in component-based development processes, component implementations must be in conformance with their respective dependent component specifications. So, in Fig. 1, note that component implementations must refer to their correspondent dependent component specifications. Besides, for each dependent component specification, several component implementations can be realized.

As an example of the description of an asset in X-ARM, Fig. 2 illustrates a fragment of a dependent component specification. In Fig. 2, all lines are numbered and many details have been suppressed for didactic purposes. Line 1 represents the asset header, in which can be found the asset identifier (id). Lines 2 to 4 make reference to the independent component specification, from which the described asset must be in conformance with. Then, lines 5 to 14 refer to all dependent interface specifications, which are provided by the described dependent component specification. Although note illustrated in Fig. 2, required interfaces can also be specified in a similar way.

```

01 <asset name="dependentCompSpec-X"
        id="compose.dependentCompSpec-X-1.0-beta">
02   <model-dependency>
03     <related-asset name="independentCompSpec-Z"
                     id="compose.independentCompSpec-Z-1.0-stable"
                     relationship-type="independentComponentSpec"/>
04   </model-dependency>
05   <component-specification>
06     <interface>
07       <provided>
08         <related-asset name="dependentInterface-A"
                           id="compose.dependentIntSpec-A-2.0-stable"
                           relationship-type="dependentInterfaceSpec"/>
09       </provided>
10      <provided>
11        <related-asset name="dependentInterface-B"
                           id="compose.dependentIntSpec-B-3.0-stable"
                           relationship-type="dependentInterfaceSpec"/>
12      </provided>
13    </interface>
14  </component-specification>
15 </asset>
```

Figure 2. Component specification in X-ARM.

IV. A CLUSTERING BASED INDEXING APPROACH

As largely recognized in the literature, the task of indexing repositories based on semi-structured data is a relevant issue [5][6][7]. One of the major challenges is to provide an indexing mechanism that reduces storage space requirements, but without excessively impacting on query processing time and precision level.

In such a context, this paper proposes a solution for optimizing the storage space required by index files. To do that, the proposed approach constructs a clustered repository, which is composed of representative assets of the set of software assets stored in the original repository. Therefore, instead of indexing the original repository, the adopted

search service ought to index the reduced set of representative assets, which make reference to the original assets. In order to identify the groups of similar assets, and, consequently, to construct the representative assets that compose each group, the paper also proposes the adoption of data clustering techniques.

Clustering techniques [9] consist of three basic phases: (i) extraction of features that express the behavior of the elements to be clustered; (ii) definition of the similarity metric in order to compare evaluated elements; and (iii) adoption of a clustering algorithm. The phase of extracting features consists in defining what information is relevant to express the evaluated element and how information is quantified. Such information defines an attribute vector and thus an element can be represented as a point in the multidimensional space. The similarity metric expresses in quantitative terms the similarity between elements. In general, a function is defined for such a purpose, in which the Euclidean distance [9] between two points (elements) is one of the more common adopted metrics. Finally, the data clustering algorithm is a heuristic that has the aim of generating groups of elements, in which each group is composed of similar elements, according to the adopted similarity metric.

A. Relevant Features

The approach proposed herein applies the clustering technique taking into account the five types of assets that can be stored in the repository, that is: dependent and independent component specifications, dependent and independent interface specifications and component implementations. The clustering technique is applied separately for each type of asset. Therefore, each type has a distinct attribute vector for representing its features.

The relevant features of an independent interface specification are its defined operations, considering their names, input and output parameters and return values. Consequently, different independent interface specifications are considered similar when they have in common a considerable subset of defined operations.

Taking into account dependent interface specifications, the relevant features are the referenced independent interface specification together with their operations. Thus, different dependent interface specifications are considered similar when they refer to the same independent interface specification or have in common a considerable subset of defined operations.

In relation to independent component specifications, for each one, the relevant feature is the set of provided independent interface specifications. So, different independent component specifications are considered similar when they have in common a considerable subset of provided independent interface specifications.

For a dependent component specification, the relevant features are its referenced independent component specification, as well as its set of provided dependent

interface specifications. Therefore, different dependent component specifications are considered similar when they refer to the same independent component specification or have in common a subset of provided dependent interfaces.

Finally, for a component implementation, the relevant feature is its referenced dependent component specification. Hence, different implementations of the same dependent component specification are considered similar.

As an example, Table I presents the attribute vector of the asset illustrated before in Fig. 2. As can be noticed, the asset is a dependent component specification. Therefore, the attribute vector is composed of its referenced independent component specification (lines 2 to 4) and its set of provided dependent interface specifications (lines 5 to 14).

TABLE I. ATTRIBUTE VECTOR OF THE ASSET X.

ID	compose.dependentCompSpec-X-1.0-beta
Independent Component Specification	compose.independentCompSpec-Z-1.0-stable
Dependent Interface Specification	compose.dependentIntSpec.A-2.0-stable compose.dependentIntSpec.B-3.0-stable

B. Similarity Metric

The similarity metric is defined based on the asset attribute vector. Since the attribute vector differs between distinct types of assets, the similarity metric is also different for each type of asset. In this approach the similarity between two assets is quantified by an integer number, called *distance*. To avoid negative distances, we defined that the initial default distance (d_i) between two assets is 300. The similarity criterion is applied and this value may decrease, in such a way that assets are considered more similar when the final distance (d_f) between them approximates to zero.

For two dependent component specifications a and b , the similarity is defined by (1), where $k(a,b) = 0$ if both assets refer to distinct independent component specifications; otherwise $k(a,b) = 200$. Let I be the number of dependent interface specifications provided by both assets and U be the set of dependent interface specifications provided by at least one of them. The term $p(a,b)$ is defined as $p(a,b) = I/U$. As can be noticed, when $p(a,b)$ is 1 both assets provide the same set of dependent interface specifications, and thus they are more similar.

$$d_f(a,b) = d_i - k(a,b) - p(a,b) \times 100 \quad (1)$$

In the case of two independent component specifications a and b , the similarity is given by (2), where $p(a,b)$ is calculated as explained before for dependent component specifications, but considering the number of independent interface specifications provided by both assets. So, let I be the number of independent interface specifications provided by both assets and U be the set of independent interface

specifications provided by at least one of them. The term $p(a,b)$ is defined as $p(a,b) = I/U$. Similarly, when $p(a,b)$ is 1 both assets provide the same set of independent interface specifications and thus, they are more similar.

$$d_f(a,b) = d_i - p(a,b) \times 300 \quad (2)$$

Analogously, for two dependent interface specifications a and b , the similarity is calculated as expressed in (3), where $l(a,b) = 0$ if both assets refer to distinct independent interface specifications; otherwise, $l(a,b) = 200$. The term $op(a,b)$ is the ratio of common operations of both assets in relation to the union of operations of these assets. Two operations are considered similar if they have the same name, the same return type and a percentage of coincidence in parameters; the value of the percentage is defined by the user.

$$d_f(a,b) = d_i - l(a,b) - op(a,b) \times 100 \quad (3)$$

Taking into account two independent interface specifications a and b , the similarity is calculated by (4), where $op(a,b)$ represents the percentage of common operations provided by both interfaces, exactly as explained before for dependent interface specifications.

$$d_f(a,b) = d_i - op(a,b) \times 300 \quad (4)$$

Finally, for two component implementations a and b , the similarity is given by (5), where $q(a,b) = 0$ if both assets refer to distinct dependent component specifications; otherwise $q(a,b) = 300$. As can be noticed, when $q(a,b)$ is 300 both assets implement the same dependent component specification, and thus they are similar.

$$d_f(a,b) = d_i - q(a,b) \quad (5)$$

As an example, consider two dependent component specifications C and D , whose attribute vectors are given in Table II and Table III, respectively. As these assets refer to distinct independent component specifications, according to (1), $k(C,D) = 0$. In this example, C and D have a common interface and together provide three different interfaces. Thus, $I = 1$, $U = 3$ and $p(C,D) = 1/3$. Hence, $d_f(C,D) = d_i - k(C,D) - p(C,D)*100 = 300 - 0 - 0.33*100 = 276.67$.

TABLE II. ATTRIBUTE VECTOR OF THE ASSET C

ID	compose.depCompSpec-C-2.0-beta
Independent Component Specification	compose.indepCompSpec-A-3.0-stable
Dependent Interface Specification	compose.depIntSpec-A-4.0-mature compose.depIntSpec-C-4.0-mature

TABLE III. ATTRIBUTE VECTOR OF THE ASSET D

ID	compose.depCompSpec-D-3.0-mature
Independent Component Specification	compose.indepCompSpec-C-3.0-stable
Dependent Interface Specification	compose.depIntSpec-B-2.0-beta compose.depIntSpec-C-4.0-mature

C. Clustering Algorithm

The proposed clustering algorithm has two stages. In the first stage, initially, assets are randomly chosen from the respective storage unit and stored in the primary memory. It is suggested to exhaust the memory capacity with this operation. Next, but still in the first stage, the classical hierarchical clustering algorithm [9] is applied to these assets. In the beginning of the algorithm each asset is considered a cluster. Then, the algorithm groups successively the two nearest clusters, until the distance between clusters is greater than an established threshold, specified by the user. The algorithm considers the similarity metric described previously to compute the distance. The combined cluster is considered a representative asset of the joined clusters. For each type of asset, the representative asset includes the relevant features for the similarity metric and also references to the joined assets. At the end of the iteration, a directory containing all formed representative assets (clusters) is stored in secondary memory.

Fig. 3 illustrates the main steps of the first stage: (a) assets are randomly selected from the repository; (b) clusters composed of similar assets are constructed by applying the hierarchical clustering algorithm; and (c) representative assets are created for representing each cluster.

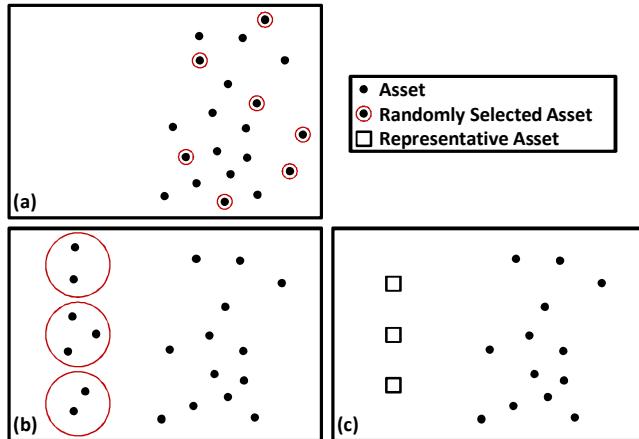


Figure 3. The first stage.

In the second stage, a K -means based algorithm [9] is adopted. In general terms, representative elements are considered centroids. However, differently from K -means, such centroids are not recalculated in the proposed approach. Indeed, each asset, not yet clustered in the first stage, is

compared with each representative asset. The asset is candidate to be included in a cluster when the distance between the asset and the respective representative asset is lesser than the threshold. Fig. 4 shows the second stage.

As depicted in Figs. 4a, 4b, and 4c, considering all candidate clusters, the asset is included in the cluster that has the minor distance and then the representative element of the cluster is reconstructed considering the features of the included asset. Otherwise, as shown in Figs. 4d, 4e and 4f, if the asset is not a candidate to any cluster, the own asset becomes a new representative element and so a new cluster.

To conclude the description of the approach, it remains to explain how the relevant features of representative assets are determined. A representative asset, resulted from the combination of two clusters composed by dependent component specifications, includes all provided dependent interface specifications of the joined assets and the independent component specification they refer. This specification is the one that mostly occurs in the assets that form the combined cluster; in the case of a draw one specification is chosen arbitrarily.

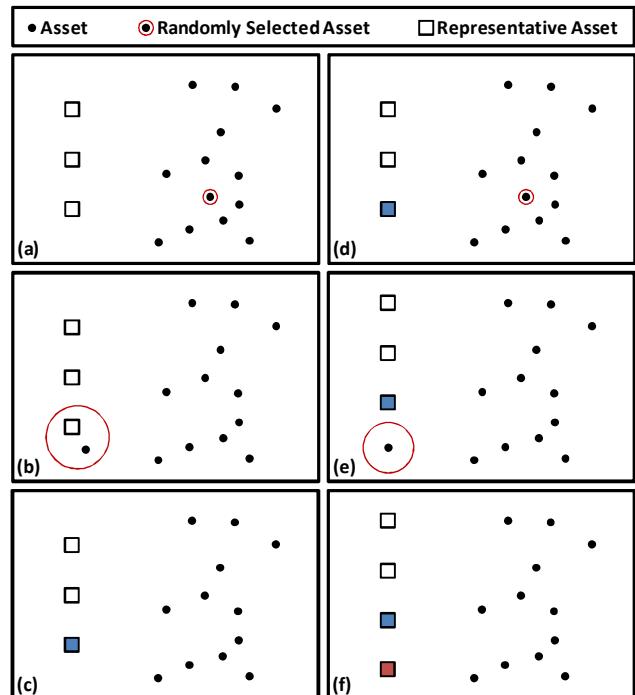


Figure 4. The second stage.

For a representative asset resulted from the combination of two clusters composed by independent component specifications, the relevant features are all provided independent interface specifications of the joined assets.

A representative asset resulted from the combination of two clusters composed by dependent interface specifications include as relevant features all operations of the joined assets, as well as the independent interface that the representative asset implements. This interface is the one mostly referred by the joined clusters.

Taking into account a representative asset resulted from the combination of two clusters composed by independent interface specification assets, the relevant features are all provided operations of the joined assets.

Finally, for a representative asset resulted from the combination of two clusters of component implementations, the relevant feature is its referenced dependent component specification. This specification is the one mostly frequent in the joined assets.

As an example of the construction of representative assets, considers two original assets as shown in Fig. 5 and Fig. 6. The representative asset resulted from the combination of these assets is described in Fig. 7. As both assets are dependent component specifications, observe that the representative asset includes all provided dependent interface specification (lines 6 to 16 of Fig. 7) and the independent component specification that occurs more frequently in the original assets. (line 3 of Fig. 7).

```

01 <asset name="depCompSpec-K"
        id="compose.depCompSpec-K-1.0-alfa">
02   <model-dependency>
03     <related-asset name="indepCompSpec-O"
                      id="compose.indepCompSpec-R-6.0-beta"
                      relationship-type="IndependentComponent"/>
04   </model-dependency>
05   <component-specification>
06     <interface>
07       <provided>
08         <related-asset name="depIntSpec-R"
                           id="compose.depIntSpec-R-3.0-mature"
                           relationship-type="dependentInterface"/>
09       </provided>
10      <provided>
11        <related-asset name="depIntSpec-S"
                           id="compose.depIntSpec-S-7.0-alfa"
                           relationship-type="dependentInterface"/>
12      </provided>
13    </interface>
14  </component-specification>
15 </asset>
```

Figure 5. Dependent component specification K.

```

01 <asset name="depCompSpec-L"
        id="compose.depCompSpec-L-2.0-pre-alfa">
02   <model-dependency>
03     <related-asset name="indepCompSpec-O"
                      id="compose.indepCompSpec-R-6.0-beta"
                      relationship-type="IndependentComponent"/>
04   </model-dependency>
05   <component-specification>
06     <interface>
07       <provided>
08         <related-asset name="depIntSpec-O"
                           id="compose.depIntSpec-O-1.0-alpha"
                           relationship-type="dependentInterface"/>
09       </provided>
10      <provided>
11        <related-asset name="depIntSpec-S"
                           id="compose.depIntSpec-S-7.0-alfa"
                           relationship-type="dependentInterface"/>
12      </provided>
13    </interface>
14  </component-specification>
15 </asset>
```

Figure 6. Dependent component specification L.

V. RESULTS AND DISCUSSION

In order to evaluate the proposed distributed clustering approach, a set of experiments has been carried out. The purpose of such experiments is three-fold. First, it is intended to identify the gains in terms of the number of representative assets to be indexed when compared with the number of original assets. The second purpose is to discover the gain in terms of storage space requirements between the clustered repository and the original repository. Lastly, such experiments have evaluated the quality of the clustering approach using well-known metrics.

```

01 <asset name="repDepCompSpec-A1"
        id="compose.repDepCompSpec-A1">
02   <model-dependency>
03     <related-asset name="indepCompSpec-O"
                      id="compose.indepCompSpec-R-6.0-beta"
                      relationship-type="IndependentComponent"/>
04   </model-dependency>
05   <component-specification>
06     <interface>
07       <provided>
08         <related-asset name="depIntSpec-O"
                           id="compose.depIntSpec-O-1.0-alpha"
                           relationship-type="dependentInterface"/>
09       </provided>
10      <provided>
11        <related-asset name="depIntSpec-R"
                           id="compose.depIntSpec-R-3.0-mature"
                           relationship-type="dependentInterface"/>
12      </provided>
13      <provided>
14        <related-asset name="depIntSpec-S"
                           id="compose.depIntSpec-S-7.0-alfa"
                           relationship-type="dependentInterface"/>
15      </provided>
16    </interface>
17  </component-specification>
18 </asset>
```

Figure 7. Representative dependent component specification.

In order to perform the experiments, it has been developed a customizable script that automatically generates a repository that stores the mentioned X-ARM assets. The generated repository has 14.000 assets of different types. After creating the repository, the proposed approach has been applied for grouping the stored assets in clusters, generating their respective representative assets.

A. Gain in Number of Assets

Fig. 8 presents the number of each type of asset in the original repository and the clustered repositories after the application of the proposed approach using different thresholds, which vary from 100 to 200 in steps of 25. As can be noticed, the proposed approach significantly reduces the number of assets. As expected, the number of resulting representative assets decreases as the threshold increases. When the threshold is increased, two assets have more chance of being considered similar, and so, more chance of being grouped together. Thus, for example, when the threshold is increased from 100 to 200, the total number of original assets is reduced to 4,287 and 2,518 representative assets, respectively.

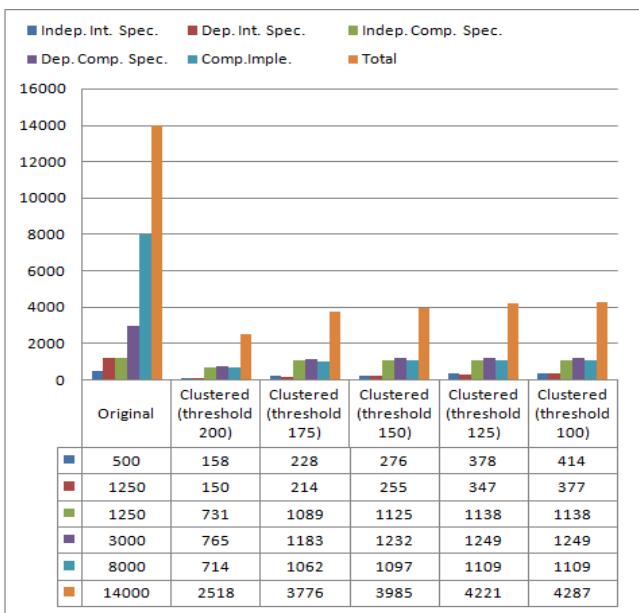


Figure 8. Number of assets.

For each considered threshold, the gain in number of assets has been identified and evaluated. Fig. 9 illustrates the gain in terms of the number of assets. For example, when the threshold is 150, the number of stored assets in the original repository is reduced around 28.5%, dropping from 14,000 original assets to 3,985 representative assets. As can be noticed in Fig. 9, the proposed approach performs a significant reduction in the number of stored assets, achieving relevant gains between 82% and 69.4%.

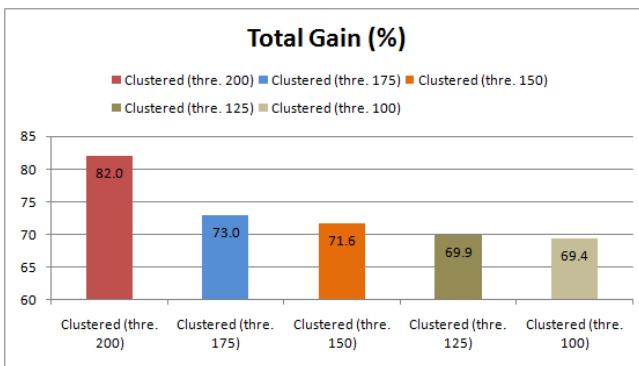


Figure 9. Total Gain in number of assets (%).

However, as shown in Fig. 10, the gains are different for each type of asset. Note that, in general, the better gains are achieved for component implementations and dependent interfaces. Considering component implementations, the gains become a little bit more expressive, varying between 91.1% and 86.1%. For dependent interface specifications, the gains are between 88% and 69.8%. In the former case, such higher gains can be explained by the considerable amount of assets of those types. As can be seen in Fig. 8, the original repository has 8,000 component implementations. Thus, this type of asset is the prevalent one in the evaluated repository,

increasing the likelihood of identifying similar assets. Furthermore, considering that component implementations are considered similar when they refer to the same dependent component implementation, it is also possible to correlate such a good gain with the existence of different implementations of the same component specification, not only for different target platforms but also for meeting a variety of non-functional requirements, like performance, security and cost. Therefore, considering the various methods, techniques and algorithms that can be employed to meet non-functional requirements, it is obvious that such multiple implementations impact on the likelihood of identifying similar component implementations.

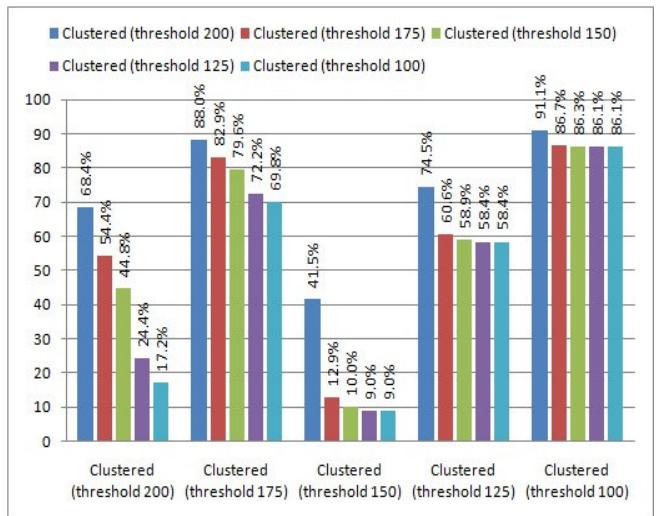


Figure 10. Gains in number of assets for different types of assets.

In the case of dependent interface specifications, the gains become better due mainly to two reasons. First, in software projects, it is not rare to implement different versions of software systems for different target platforms. So, in component-based software projects, different versions imply on several dependent interface specifications for each independent interface specification. Considering that dependent interface specifications are considered similar when they refer to the same independent interface specification, it is easy to see that multiple implementations impacts on the likelihood of identifying similar dependent interface specifications. The second reason is a consequence of the high gains in independent interface specifications. For instance, consider two dependent interface specifications ($depInt_i$ and $depInt_j$) that refer to two independent interface specifications ($indepInt_x$ and $indepInt_y$), respectively. Now, consider that $indepInt_x$ and $indepInt_y$ are clustered as the representative asset $indepInt_c$. As a consequence, now, both dependent interface specifications $depInt_i$ and $depInt_j$ refer to the same representative independent interface specification $indepInt_c$. Then, taking into account that dependent interface specifications are considered similar when they refer to the same independent interface specification, $depInt_i$ and $depInt_j$ are clustered and produce the representative asset $depInt_c$.

Clearly, a high gain in clustering independent interfaces has a significant impact in the gain in clustering dependent interfaces.

In terms of dependent component specifications, the gains range from 74.5% to 58.4%. One reason for this gain is the expressive number of assets in the repository (3,000 assets according Fig. 8). Furthermore, the existence of different versions of software systems for different target platforms implies on several dependent component specifications for each independent component specification. Considering that dependent component specifications are considered similar when they refer to the same independent component specification, it is clear to notice that multiple implementations impacts on the likelihood of identifying similar dependent component specifications.

In relation to independent component specifications, the gains are notably low, varying from 41.5% to 8.9%. Besides, as can be noticed in Fig. 10, the gain of 41.5% occurs for the higher threshold only. When the threshold is 175 and 125, the respective gains decrease to 12.9% and 8.9%. Such gains are relatively low and indeed not expected. As mentioned before, independent component specifications are considered similar when they have in common a considerable subset of provided independent interfaces. Thus, it is possible to infer that such low gains are a consequence of the difficulty of finding two or more independent component specifications that share a reasonable subset of independent interfaces.

As can be noticed, the clustering gains in independent interfaces specifications impact positively on the gains in dependent interface specifications, but give the impression that do not impact on the gains in independent component specifications. Furthermore, the clustering gains in independent component specifications impact on the gains in dependent component specifications, which in turn impact on the gains in component implementations.

B. Gain in Storage Requirements

As already mentioned, the adoption of semi-structured data for representing metadata about software components implies challenges related to the design of indexing techniques that ought to be efficient in terms of storage space requirements. Therefore, it is not enough to be efficient in reducing the number of assets, but also in downgrading storage space requirements for index files.

In such a direction, the gain in terms of storage space required by index files has been evaluated in the original repository, containing 14.000 X-ARM assets of different types. After generating the clustered repositories by applying the proposed approach for different thresholds, the original repository and the clustered repositories have been indexed using the indexing technique proposed in [7]. Fig. 11 presents the storage space required by the original repository and the clustered repositories, after applying the indexing technique.

As can be noticed, the proposed approach significantly reduces the required storage space. As expected, the required

storage space decreases as the threshold increases. When the threshold increases, the number of representative assets reduces, and, as a consequence, the storage space required by index files also downgrades. Thus, when the threshold increases from 100 to 200, the storage space required by index files reduces from 10.9 to 7.3MB.

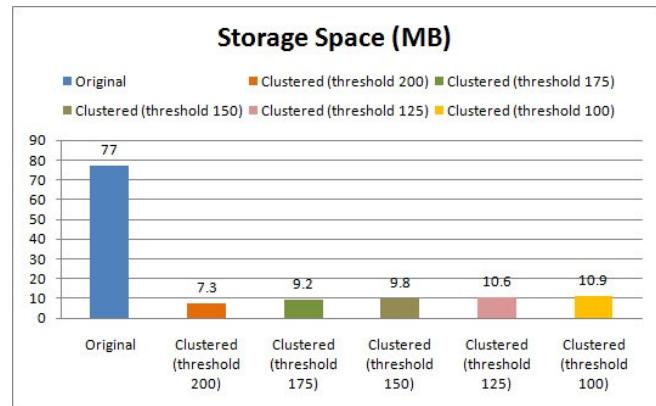


Figure 11. Storage space requirements for different thresholds.

For each considered threshold, the gain in storage space requirements has been identified and evaluated. Fig. 12 illustrates the gain in terms of storage space requirements. For example, when the threshold is 150, the storage space required by index files is reduced around 87.3%, dropping from 77 to 9.8 MB. As can be noticed in Fig. 12, the proposed approach performs a significant reduction in the storage space requirements, achieving relevant gains between 85.8% and 90.5%.

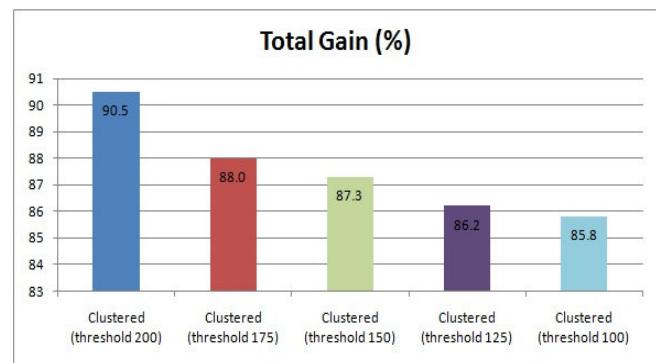


Figure 12. Total gain in storage requirements.

C. Clustering Quality

Of course, it is not enough to evaluate the gains in terms of number of assets and storage space requirements. It is also imperative to assess the quality of the clustering approach. In such a direction, the clustered repositories have been assessed in terms of two different clustering evaluation methods: *Davies-Bouldin index* and *Silhouette index*.

The Davies-Bouldin index [19] is a clustering evaluation method based on internal criterion. It is a function of the ratio of the sum of intra-cluster distances (within-cluster scatter) to inter-cluster distances (between-cluster

separation), as defined in (6), where n is the number of clusters, c_i is the representative element of cluster i , σ_i is the average distance of all elements in cluster i to representative element c_i , and $d(c_i, c_j)$ is the distance between representative elements c_i and c_j . As widely mentioned in the literature, a good clustering algorithm must produce clusters with low intra-cluster distances (high intra-cluster similarity) and high inter-cluster distances (low inter-cluster similarity). Based on that, a good clustering algorithm has a small value of Davies-Bouldin index, representing compact and well-separated clusters [20].

$$DB = \frac{1}{n} \sum_{i=1}^n \max_{i \neq j} \left\{ \frac{\sigma_i + \sigma_j}{d(c_i, c_j)} \right\} \quad (6)$$

As can be seen in Fig. 13, the Davies-Bouldin index of the clustered repositories for all evaluated thresholds varies between 9.60 and 1.66. Such low values for the threshold from 100 to 150 evinces that the proposed approach can be considered a good clustering algorithm because has produced compact and well-separated clusters. Note that the Davies-Bouldin index increases as the threshold increases. Such a trend is already expected and indicates that lower thresholds produce higher intra-cluster similarity and lower inter-cluster similarity, and so higher quality clusters.

The Silhouette index [21] is based on the comparison of the tightness and separation of the clustered elements. The silhouette for each element is calculated as illustrated in (7), where a_i is the average intra-cluster dissimilarity of element i to all other elements within the same cluster, and b_i is the lowest average inter-cluster dissimilarity of element i to all other elements in another cluster. Note that the silhouette value varies between -1 and 1. Based on the silhouette for each element, the overall average silhouette for all elements can be easily calculated. If the overall average silhouette is close to 1, it means that elements are well-clustered and are assigned to very appropriate clusters. If the overall average silhouette is close to -1, it means that elements are misclassified and so poorly clustered.

$$S(i) = \frac{b_i - a_i}{\max\{a_i, b_i\}} \quad (7)$$

As also illustrated in Fig. 13, the overall average silhouette of the clustered repositories for all evaluated thresholds varies between 0.62 and 0.84. Such values close to 1 are evidences that the proposed approach is a good clustering algorithm because elements are well-clustered and are assigned to appropriate clusters. Note that the silhouette index decreases as the threshold increases. Again, such a trend is already expected and indicates that lower thresholds produce lower intra-cluster dissimilarity and higher inter-cluster dissimilarity, and so higher quality clusters.

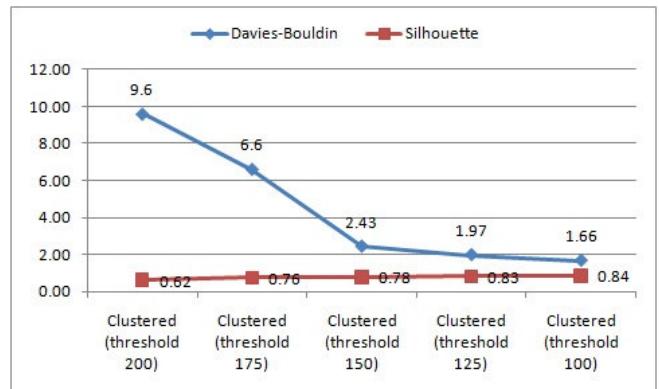


Figure 13. Quality indexes for different thresholds.

VI. CONCLUSION

Based on the preliminary results, it can be clearly evinced as benefits the potential of the proposed approach in significantly clustering an X-ARM repository and consequently reducing storage space requirements. It must be highlighted that, the bigger the original repository in terms of the number of stored assets, the more expressive the likelihood of clustering assets, and so the better the gain in terms of storage space requirements.

Taking into account that the indexing technique proposed by Brito *et al.* [7] adopted for indexing the clustered repository, the experiments reveal the reduction in the size of the original repository implies in an expressive reduction in the size of index files of the clustered repository. Besides, considering that the technique proposed by Brito *et al.* has an excellent performance in query processing time, even in large-scale index files, it is expected a reasonable gain in terms of query processing time due to the expressive reduction in the size of index files. Therefore, the proposed approach clearly makes possible to map large software component repositories into small index files.

However, as often informally said, there is no free lunch. That is, in formal words, such expressive gains in terms of storage space requirements and query processing time have an impact on the query precision level, since the process of clustering assets introduces some degree of information loss in representative assets. For the experiments of the previous section the query precision level vary from 0.41 for the threshold of 100 to 0.31 for the threshold of 200. Such results can be considered very attractive because, as indicated in experiments presented in [22], highly popular and adopted search engines like Google and Altavista have achieved inferior precision indexes around 0.29 and 0.27, respectively. Moreover, in all thresholds the recall index is about 0.67. Again, such results can also be considered interesting because, as also indicated in [22], Google and Altavista have obtained inferior recall indexes around 0.20 and 0.18, respectively.

Although these preliminary results indicate the usefulness of the approach, a large number of experiments must be performed to better evaluate the heuristics and the

similarity metric here introduced. In these experiments we must investigate several configurations of the repository differing on the amount of assets of each type, as well as the possibilities of relations among them. Besides, it is also under investigation a comparative analysis contrasting the proposed heuristics and other ones available in the literature, but applied in different research fields.

ACKNOWLEDGMENT

This work was supported by the National Institute of Science and Technology for Software Engineering (INES – www.ines.org.br), funded by CNPq, grants 573964/2008-4.

REFERENCES

- [1] A. Orso, M.J. Harrold, and D.S. Rosenblum, "Component Metadata for Software Engineering Tasks", Proc. 2nd Int. Workshop on Engineering Distributed Objects, 2000, pp. 126-140.
- [2] P. Vitharana, F. Zahedi, and H. Jain, "Knowledge-Based Repository Scheme for Storing and Retrieving Business Components: A Theoretical Design and an Empirical Analysis", IEEE Transactions on Software Engineering., vol. 29, issue 7, July 2003, pp. 649-664.
- [3] OMG, Reusable Asset Specification: OMG Available Specification – v2.2, 2005.
- [4] G. Elias, M. Schuenck, Y. Negócio, J. Dias, and S. Miranda, "X-ARM: An Asset Representation Model for Component Repository", Proc. 21st ACM Symposium on Applied Computing (SAC 2006), France, 2006, pp. 1690-1694.
- [5] W. Meier, "eXist: An Open Source Native XML Database", NODe 2002 Web and Database-Related Workshops on Web, Web-Services, and Database Systems, 2002.
- [6] R. Goldman and J. Widom, "DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases", Proc. 23rd Int. Conf. on Very Large Data Bases (VLDB 1997), Greece, 1997, pp. 436-445.
- [7] T. Brito, T. Ribeiro, and G. Elias, "Indexing Semi-Structured Data for Efficient Handling of Branching Path Expressions", 2nd Inter. Conf. on Advances in Databases, Knowledge, and Data Applications (DBKDA 2010), France, 2010, pp. 197-203.
- [8] M.P. Paixão, L. Silva, T. Brito and G. Elias, "Large Software Component Repositories into Small Index Files", Proc. 3rd International Conference on Advances in Databases, Knowledge, and Data Applications (DBKDA 2011), pp. 122-127, 2011.
- [9] A.K. Jain and R.C. Dubes, Algorithms for Clustering Data, Prentice Hall, 1984.
- [10] R. Xu and D. Wunsch, "Survey of Clustering Algorithms", IEEE Transactions on Networks, vol.16, issue 3, May, pp. 645-678.
- [11] A. Feng, "Document Clustering – An Optimization Problem", ACM SIGIR 2007, pp. 819-820.
- [12] S. Mancoridis, B.S. Mitchell, C. Rorres, Y.F. Chen, and E.R. Gansner, "Using automatic clustering to produce high level system organizations of source code". Proc. IEEE International Workshop on Program Comprehension, pp. 45–53, 1998.
- [13] B.S. Mitchel and S. Mancoridis. "On the Automatic Modularization of Software Systems Using the Bunch Tool", IEEE Transaction on Software Engineering, vol. 32, issue 3, pp. 1-16, March 2006.
- [14] Y. Chiricota, F. Jourdan, and G. Melançon, "Software Component Capture using Graph Clustering", Proc. IEEE International Workshop on Program Comprehension, 2003.
- [15] J. Wu, A.E. Hassan, and R.C. Holt, "Comparison of Clustering Algorithms in the Context of Software Evolution", Proc. 21st Int. Conf. on Software Maintenance, 2005, pp. 525-535.
- [16] Z. Li, Q.A. Rahman, and N.H. Madhavji, "An Approach to Requirements Encapsulation with Clustering", Proc. 10th Workshop on Requirement Engineering, 2007, pp. 92-96.
- [17] M. Cohen, S.B. Kooi, and W. Srisa-an. "Clustering the Heap in Multi-Threaded Applications for Improved Garbage Collection", Proc. of the 8th annual Conference on Genetic and Evolutionary Computation, Vol. 2, pp. 1901-1908, July 2006.
- [18] W. Frakes and K. Kang, "Software Reuse Research: Status and Future", IEEE Transactions on Software Engineering, vol.31, issue 7, July 2005, pp. 529-536.
- [19] D.L. Davies, and D.W. Bouldin, "A Cluster Separation Measure", IEEE Trans. Pattern Anal. Mach. Intelligence, vol. 1, pp. 224–227, 1979.
- [20] S. Theodoridis and K. Koutroumbas, Patternn Recognition, Academic Press, 2009.
- [21] P.J. Rousseeuw, "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis", Journal of Computational and Applied Mathematics, vol 20, pp. 53-65, 1987.
- [22] S.M. Shafi and R.A. Rather, "Precision and Recall of Five Search Engines for Retrieval of Scholarly Information in the Field of Biotechnology", Webology, vol. 2, number 2, 2005.

Superposition of Rectangles with Visibility Requirement: A Qualitative Approach

Takako Konishi

*Graduate School of Science & Technology
Kwansei Gakuin University
2-1, Gakuen, Sanda, 669-1337, JAPAN
Email: t.konishi@kwansei.ac.jp*

Kazuko Takahashi

*School of Science & Technology
Kwansei Gakuin University
2-1, Gakuen, Sanda, 669-1337, JAPAN
Email: ktaka@kwansei.ac.jp*

Abstract—This paper discusses the superposition of qualitative rectangles so that some parts are visible and other parts are hidden based on the user's requirements. Qualitative rectangles are rectangles whose size and edge ratios are not fixed. We propose a symbolic representation of the target objects and discuss superposition on this representation. The operations of superposing two rectangles can be defined either by superposing some specified parts of rectangles or by embedding one rectangle into part of the other rectangle. We investigate the conditions under which such a superposition succeeds as well as the manner in which such superposition occurs. We developed an algorithm for superposing multiple qualitative rectangles and implemented it.

Keywords-qualitative knowledge representation; superposition; rectangle packing; spatial database

I. INTRODUCTION

Personal computer users commonly open multiple windows. When many windows are opened on a narrow screen, the most newly opened window usually appears in the foreground, sometimes hiding important parts of previously opened windows. Users must frequently resize windows or move a window to the foreground to ensure that important parts are visible. Many users find this process annoying; it would be more convenient if windows were positioned automatically so that important or necessary parts remain visible under the condition that these parts are specified in advance. Moreover, considering the limited screen space, it would be more efficient if less important parts of windows are hidden.

In general, researchers have investigated the efficient placement of objects as a type of packing problem for which an optimal solution can be determined [1]. They have focused on several application areas, such as VLSI design [2] and label-placement problems [3], [4]. The objective of these studies has been to determine how multiple objects are located in a two-dimensional plane under circumstances not involving superposition, which differs from our objective of determining placement involving superposition. Therefore, we cannot directly apply the previously developed algorithms for general placement problems. To the best of the authors' knowledge, no study has been performed on the location of objects involving superposition.

In this study, we discuss rectangle placement with superposition. We treat rectangles using qualitative representation: their sizes and the ratios of their edges are unfixed. In each rectangle, the desired visible part is specified. We discuss a manner of superposing them so that all desired visible parts are in the foreground and all desired hidden parts are in the background. Figure 1 illustrates several examples. Assume that three rectangles A, B, and C are given with a requirement of visibility specified by a user. The white indicates the parts that one wants to be visible, and the black indicates the parts that one wants to be hidden. In this figure, (a), (b), and (c) are successful cases, whereas (d) is not. In (c), first reduce B's width to fit the vertically-long-size subpart of the black part of A, then C is put on the black part in the lower left part of the resultant figure. In (d), visible black parts remain after superposing A and B cannot be hidden by C in any superposition of A and B. In this paper, we show how we evaluate the success of superposition and placement in these cases.

Note that the sizes or ratios of edges can change during superposing. We take a qualitative approach. One reason for this is that it enables symbolic handling of objects. In general, spatial data can be inconveniently large to store and handle. Symbolic handling reduces this computational complexity. Another reason is that it is enough to know the relative positional relationship of objects on a two-dimensional plane and their foreground/background relationship, ignoring the exact size or position of each object. Such an idea is considered to be a type of qualitative spatial reasoning (QSR) in the field of artificial intelligence [5], [6], [7], [8].

Our goal is not to find an optimal solution to the packing problem, but to investigate methods for symbolic treatment of spatial data and to develop possible application areas of qualitative spatial reasoning. An earlier version of our work was reported in [9], [10]. The present study expanded on our previous research, and this paper provides a more detailed discussion.

In our QSR approach, target rectangles are divided into nine types depending on the specified visibility pattern. We define a unique symbolic representation for each type and investigate superposition operations using these representa-

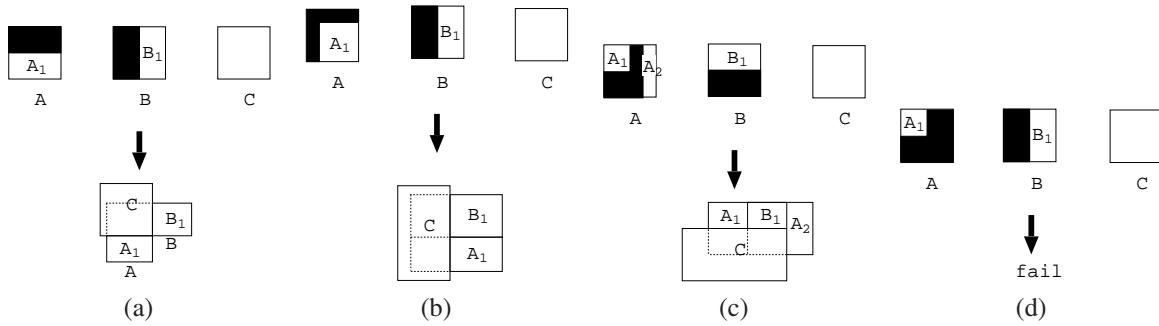


Figure 1. Examples of superposing rectangles

tions. Superposition succeeds if any rectangle is not placed on the desired visible parts of another rectangle.

We focus on two types of superposition. *puton* is an operation that corresponds to superposition of specified parts of two rectangles. *embed* is an operation that corresponds to the embedding of a whole part of one rectangle into the other rectangle. *puton* is defined as a function for a pair of symbolic representations. We show the conditions for success of *puton*. *embed* cannot be defined on the symbolic representation as *puton*, but it generates a solution for superposition that cannot be generated by *puton*. We explain these two operations, discuss their properties, and compare them. We also present an algorithm for superposing multiple rectangles using these two operations.

This paper is organized as follows. In Section II, we briefly explain qualitative spatial reasoning, which is the foundation of our approach. In Section III, we define the target object and describe its qualitative representation. In the following two sections, we describe the operations for superposing a pair of qualitative rectangles, and discuss our reasoning regarding superposition. In Section IV, we discuss the *puton* operation, and in Section V, we discuss the *embed* operation. In Section VI, we describe an algorithm for superposing multiple rectangles and show a behavior of an implemented system. Finally, in Section VII, we present our conclusions.

II. QUALITATIVE SPATIAL REASONING

Qualitative spatial reasoning (QSR) is a method for representing an objective spatial entity qualitatively, rather than using exact numeral data, and for reasoning about the properties that hold on these data [6], [7]. It extracts only the necessary aspects of the objective spatial data and represents them symbolically. For example, quantitative representation of Figure 2 is as follows: "There are two objects: one is a rectangle whose nodes at the bottom left are (1,1) and the length of the two edges are 3 and 5; the other is a circle whose center is (5,7) and radius is 2." However, the figure can also be qualitatively represented as follows: "There are two objects that have a common part." This is sufficient information for a discussion of the positional relationships

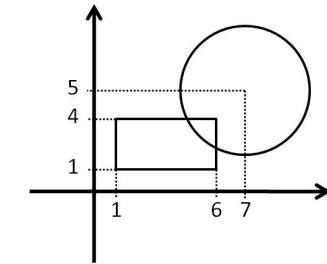


Figure 2. Qualitative versus quantitative representation

of objects. Moreover, if these objects are moving in time, their positional relationships changes. In some applications, we only need to focus on the instant in which disconnected objects change to become connected or in which connected objects change to have a relationship of inclusion, ignoring the exact distance between them or the exact size of their intersection part. QSR can provide a simple representation and reduce computational complexity.

Various QSR calculi or systems are available depending on what aspects of spatial data are interested, such as positional relationship, direction, distance, size, orientation or shape. Several studies have focused on qualitative spatial databases. For example, Wang and Liu developed a QSR application for a geospatial semantic web by constructing a qualitative spatial database that stores objects and their qualitative relations instead of coordinates, from the Geography Markup Language (GML) [11]. Santos and Amaral proposed an approach to develop a qualitative database by introducing qualitative identifiers such as direction and relative distance and applied it to data mining [12]. Although these studies have shown the effectiveness of qualitative spatial databases, further studies are required. Applications of QSR include geographic information systems, robot planning, navigation, and spatial databases, but few concrete applications have been developed to date.

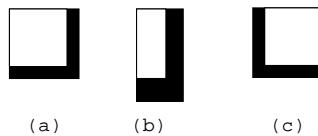


Figure 3. Qualitative rectangles

III. DESCRIPTION

First, we give a symbolic representation to the target objects.

We call a superposing entity *a unit*. A unit is a rectangle and is divided into WHITE, which should be visible, and BLACK, which should be hidden. BLACK is divided into a *core region* and a *non-core region*, which will be defined later. The outer side of a unit is called GRAY. The length of edges and the ratios of a unit and of each region are unfixed. In contrast, the orientation of a unit should be fixed. We only use rectangles situated in an upright position and do not consider those in an inclined orientation. This means that (a) and (b) in Figure 3 are regarded as equivalent, while (a) and (c) are regarded as different.

Each connected WHITE is called *a white region*, the core region and connected non-core regions are called *black regions*, and GRAY is called *a gray region*. White, black, and gray regions have attribute values related to visibility, denoted by 'w', 'b', and 'g.' 'w' and 'b' denote that regions should be visible and hidden, respectively. 'g' denotes that there is no requirement with respect to visibility.

Considering the structure of web page frames or the style of dividing a window into sub-windows used in many applications, we restricted the type of unit to those in Figure 6.

Any unit can be defined as a qualitative rectangle using the following operation that fits black plates into a white rectangle. Conversely, a qualitative rectangle obtained by this operation is only the units shown in Figure 6. Let $a * b$ represent a size of a unit whose length is a and height is b . Consider two black plates whose sizes are $x * b$ ($0 \leq x \leq a$) and $a * y$ ($0 \leq y \leq b$). Fit these plates into a white rectangle while preserving their orientation using either of the following procedures. Symbols enclosed in parentheses denote the names of unit types.

- (0) No plate is fit (W).
- (1) Only one of the plates is fit (B, I1, I2).
- (2) Both plates are fit (L1, T1, PLUS).
- (3) Extend L1 and T1, respectively, where the white region is added to the part on which the edge of size a or b is connected to the outer part (L2, T2).

Definition 1. The unit obtained in this manner is said to be valid.

The following theorem clearly holds.

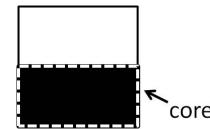


Figure 4. Core region and non-core region of straight-plate-unit

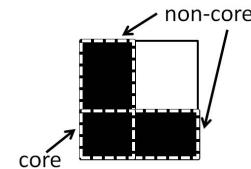


Figure 5. Core region and non-core region of cross-plates-unit

Theorem 2. A unit is valid iff (i) the whole shape is rectangular, (ii) it has one connected BLACK, and (iii) all its white regions are convex.

Types I1 and I2 are called *straight-plate-units*. Types L1, L2, T1, T2, and PLUS are called *cross-plates-units*.

For all units other than the W-type unit, the *core region* is defined. For B-type and straight-plate-units, the core region is defined as the entire BLACK (Figure 4). For cross-plates-units, the core region is defined as the intersection of the two plates, and the region not included in the core region is called the *non-core region* (Figure 5).

We denote the core region of a unit X by Core_X . The valid unit can be uniquely represented as a quadruple of attribute values composed of Core_X 's upper region, right region, lower region, and left region. We call this a *representation for a unit*. For example, the representation for the unit in Figure 5 is $\langle b, b, g, g \rangle$ because the core region has black regions in its upper side and right side, whereas it is connected to the outside in its lower side and left side. Note that the positional relationships of regions are preserved even if the size of a unit is changed.

Let V, R ($R \subset V^4$), and T indicate a set of attribute values, a set of representations for units, and a set of types, that is:

$$\begin{aligned} V &= \{b, w, g\} \\ R &= \{(r_1, r_2, r_3, r_4) \mid \text{a representation for a valid unit}\} \\ T &= \{\text{'B', 'W', 'I1', 'I2', 'L1', 'L2', 'T1', 'T2', 'PLUS'}\} \end{aligned}$$

The function $\text{rotate}(r)$, which denotes a $\pi/2$ clockwise rotation of a unit r , and the function $\text{symm}(r)$, which denotes a symmetric transformation of a unit r , are defined as follows:

Let r be $\langle r_1, r_2, r_3, r_4 \rangle$.

$\text{rotate} : R \rightarrow R$

$$\text{rotate}(\langle r_1, r_2, r_3, r_4 \rangle) = \langle r_2, r_3, r_4, r_1 \rangle$$

$\text{symm} : R \rightarrow R$

$$\text{symm}(\langle r_1, r_2, r_3, r_4 \rangle) = \langle r_1, r_4, r_3, r_2 \rangle$$

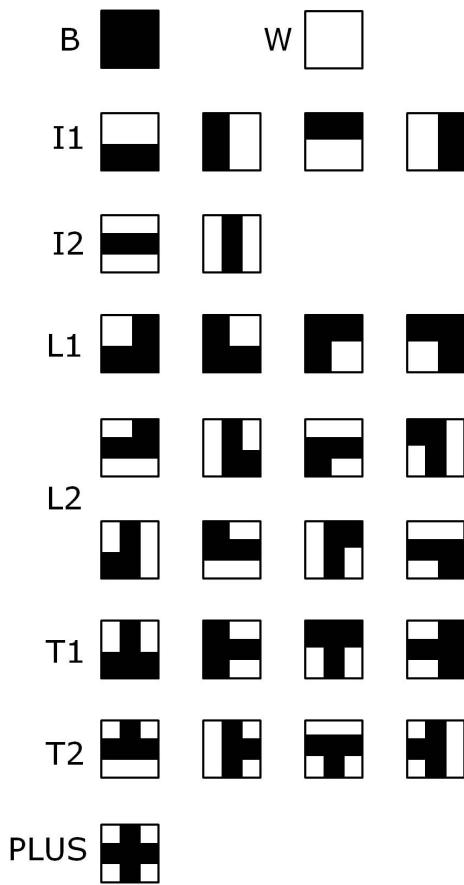


Figure 6. All valid units

The function *type* that defines the type for a representation $r \in R$ is defined as follows:

$$\begin{aligned}
 \text{type} : R &\rightarrow T \\
 \text{type}((g, g, g, g)) &= 'B' \\
 \text{type}((w, w, w, w)) &= 'W' \\
 \text{type}((w, g, g, g)) &= 'I1' \\
 \text{type}((w, g, w, g)) &= 'I2' \\
 \text{type}((b, g, g, b)) &= 'L1' \\
 \text{type}((b, g, w, b)) &= 'L2' \\
 \text{type}((b, b, g, b)) &= 'T1' \\
 \text{type}((b, b, w, b)) &= 'T2' \\
 \text{type}((b, b, b, b)) &= 'PLUS'
 \end{aligned}$$

For representations $r, r' \in R$, if $r' = \text{rotate}(r)$ or $r' = \text{symm}(r)$ holds, then $\text{type}(r')$ is defined as $\text{type}(r)$.

Note that W-type is defined with the assumption that a tiny core region exists and is surrounded by white regions, as Core_X does not exist.

The projections from $r \in R$ to its elements are defined as follows:

$$\text{up/dn/lr/rt} : R \rightarrow V$$

Let r be $\langle r_1, r_2, r_3, r_4 \rangle$.

$$\begin{aligned}
 \text{up}(r) &= r_1 \\
 \text{rt}(r) &= r_2 \\
 \text{dn}(r) &= r_3 \\
 \text{lr}(r) &= r_4
 \end{aligned}$$

IV. REASONING ABOUT SUPERPOSITION: PUTON

A. The principle

When n ($n \geq 3$) units are given, we consider the manner of their superposition in which all white regions are visible and all black regions are hidden.

Here, we place units sequentially. k -th unit ($n \geq k \geq 2$) should be placed on the figure composed of $k - 1$ units so that at least one region of the former is placed on at least one region of the latter. That is, we do not consider the placement in which, after two units are placed in a disconnected manner, a third unit is placed onto the black region of the two rectangles simultaneously. Thus, there should be at least one W-type unit. Here, we assume that there is one W-type unit. When more than one W-type unit exists, the scenario can be considered similarly. Then, the only one connected rectangular BLACK should be visible when superposition of $n - 1$ units is completed.

There are only two operations, *puton* and *embed*. *puton* is an operation of superposing the core regions of two valid units, whereas *embed* is an operation of superposing the whole unit onto a part of another unit. We describe these operations in detail.

B. Superposing the core regions

First, we describe *puton* operation.

Definition 3. Suppose that a straight-plate-unit Y is put on a unit X . Let Core_X and Core_Y be the core regions of X and Y , respectively. The superposition in which Core_Y is placed exactly on Core_X is called *puton* operation.

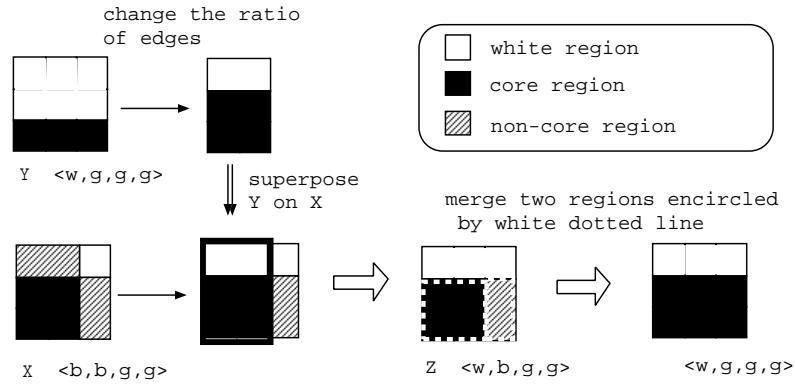
Let Z be the resultant figure of *puton*, and let Core_Z be the superposed region of Core_X and Core_Y . We extend a representation for a unit to be available as a representation for Z . A representation for Z is a quadruple of the attribute values of visible regions surrounding Core_Z .

First, we compute the attribute values of the regions around Core_Z . We define the function *on*, which computes the attribute value of the visible region when the second region is placed exactly on the first region, from the attribute values of the two regions.

$$\begin{aligned}
 \text{on} : V \times V &\rightarrow V \cup \{\text{fail}\} \\
 \text{on}(b, b) &= b \\
 \text{on}(b, w) &= w \\
 \text{on}(w, w) &= \text{fail} \\
 \text{on}(w, b) &= \text{fail} \\
 \text{on}(g, v) &= v \text{ where } v \in V \\
 \text{on}(v, g) &= v \text{ where } v \in V
 \end{aligned}$$

'fail' means that the operation failed in that case.

When the result is not *fail*, X 's black regions are sometimes visible in Z . If they are connected with Core_Z

Figure 7. A case in which *merge* is necessary

by lines, it is necessary to merge them to define the merged region as a new $Core_Z$. For example, in Figure 7, X and Y are represented as $\langle b, b, g, g \rangle$ and $\langle w, g, g, g \rangle$, respectively. When we place Y on X such that $Core_Y$ is placed on $Core_X$, the resultant figure Z is represented as $\langle w, b, g, g \rangle$. X 's non-core region is visible and is connected with $Core_Z$ by a line. Then, this region is merged with $Core_Z$. This function *merge* is defined as follows:

Let $r = \langle r_1, r_2, r_3, r_4 \rangle$. If r satisfies $\bigwedge_{i=1, \dots, 4} (r_i \neq fail)$, then *merge* can be defined.

$$\text{merge} : V^4 \rightarrow R$$

$$\text{merge}(r) =$$

$$\left\{ \begin{array}{ll} \langle g, r_2, g, r_4 \rangle & \text{if } r_1 = b \wedge r_2 \neq b \wedge r_3 = b \wedge r_4 \neq b \\ \langle r_1, g, r_3, g \rangle & \text{if } r_1 \neq b \wedge r_2 = b \wedge r_3 \neq b \wedge r_4 = b \\ \langle g, r_2, r_3, r_4 \rangle & \text{if } r_1 = b \wedge r_2 \neq b \wedge r_3 \neq b \wedge r_4 \neq b \\ \langle r_1, g, r_3, r_4 \rangle & \text{if } r_1 \neq b \wedge r_2 = b \wedge r_3 \neq b \wedge r_4 \neq b \\ \langle r_1, r_2, g, r_4 \rangle & \text{if } r_1 \neq b \wedge r_2 \neq b \wedge r_3 = b \wedge r_4 \neq b \\ \langle r_1, r_2, r_3, g \rangle & \text{if } r_1 \neq b \wedge r_2 \neq b \wedge r_3 \neq b \wedge r_4 = b \\ \langle r_1, r_2, r_3, r_4 \rangle & \text{otherwise} \end{array} \right.$$

Success of puton operation

For valid units X and Y whose representations are $r = \langle r_1, r_2, r_3, r_4 \rangle$ and $r' = \langle r'_1, r'_2, r'_3, r'_4 \rangle$, respectively, the *puton* operation that puts Y on X is defined as follows and succeeds if (c1) holds.

$$\text{puton} : R \times R \rightarrow R$$

$$\text{puton}(r, r') =$$

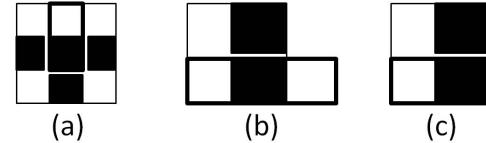
$$\text{merge}(\langle on(r_1, r'_1), on(r_2, r'_2), on(r_3, r'_3), on(r_4, r'_4) \rangle)$$

$$(c1) \quad \bigwedge_{i=1, \dots, 4} on(r_i, r'_i) \neq fail.$$

When the *puton* operation succeeds, it results in a superposition in which no white region or black region is put on a white region, and the following property clearly holds due to the definition of *puton*.

Theorem 4. If the *puton* operation succeeds, BLACK of the resultant figure is connected.

When the *puton* operation succeeds, it produces figures such as Figure 8. (a) is the result of putting I1-type unit on

Figure 8. Resultant figures when *puton* succeeds

PLUS-type unit, that is, $\text{puton}(\langle b, b, b, b \rangle, \langle w, g, g, g \rangle)$. The result is $\langle w, b, b, b \rangle$. (b) is the result of putting I2-type unit on L1-type unit, that is, $\text{puton}(\langle b, g, g, b \rangle, \langle g, w, g, w \rangle)$. The result is $\langle g, w, g, w \rangle$. And (c) is the result of putting I1-type unit on L1-type unit, that is, $\text{puton}(\langle b, g, g, b \rangle, \langle g, g, g, w \rangle)$. The result is $\langle g, g, g, w \rangle$. When superposing multiple units, we superpose another rectangle on these figures. In this case, these figures should satisfy two more conditions for continue superposition: *effectiveness* and *validity*.

Let Z be the resultant figure of superposing X and Y .

Definition 5. Z has only one connected BLACK that is visible and rectangular, then Z is said to be effective.

Definition 6. If Z 's entire shape is rectangular and all of its white regions are convex, then Z is said to be valid.

From Theorem 4, if Z is valid, then Z is a valid unit.

When $n - 1$ units are superposed, the resultant figure should have only one connected visible BLACK, which is finally hidden by placing the W-type unit. This explains why the resultant figure of *puton* should be effective. For example, Figure 8(a) is not effective. Moreover, the figure obtained as intermediate data in the process of superposing $n - 1$ units should be valid for the following continuous superposition. For example, Figure 8(b) is not valid. Figure 8(c) is both effective and valid. The conditions of effectiveness and validity can be checked using the following rules.

Effectiveness

Let r be a representation for Z . If r satisfies (c2), then

Z 's BLACK is rectangular.

$$(c2) \quad \bigwedge_{i=1, \dots, 4} (r_i \neq b),$$

Validity

Let $r = \langle r_1, r_2, r_3, r_4 \rangle$, $r' = \langle r'_1, r'_2, r'_3, r'_4 \rangle$ and $r'' = \langle r''_1, r''_2, r''_3, r''_4 \rangle$ be representations for units X , Y and Z , respectively. For the entire shape of Z to be rectangular, the white region of Y should not be placed on GRAY of X . Moreover, all of Z 's white regions are convex. Therefore, if (c3) and (c4) are satisfied, then Z is valid. In the followings, r_i is regarded as r_{i-4} when $i \geq 5$.

$$(c3) \quad \text{If there exists } i \ (1 \leq i \leq 4) \text{ such that } r_i = r'_{i+2} = g \text{ and that satisfies one of the followings:}$$

- (i) $r_{i+1} = b \wedge r'_{i+1} \neq g \wedge r_{i+3} = g \wedge r'_{i+3} = g$
- (ii) $r_{i+1} = g \wedge r'_{i+1} = g \wedge r_{i+3} = b \wedge r'_{i+3} \neq g$
- (iii) $r_{i+1} = b \wedge r'_{i+1} \neq g \wedge r_{i+3} = b \wedge r'_{i+3} \neq g$
- (iv) $r_{i+1} = g \wedge r'_{i+1} = g \wedge r_{i+3} = g \wedge r'_{i+3} = g$

$$(c4) \quad \text{No } i \ (1 \leq i \leq 4) \text{ exists such that satisfies either of the followings.}$$

- (i) $r''_i = r''_{i+1} = b$
- (ii) $r''_i = r''_{i+1} = r''_{i+2} = w$
- (iii) $(r''_i \neq g) \wedge (r''_{i+2} \neq g) \wedge (r''_{i+1} = b)$

C. Result of superposition: puton

In Definition 3, we defined the *puton* operation for the superposition of a straight-plate-unit and a unit. In this subsection, we extend this operation to any pair of unit types. We also discuss the effectiveness and validity of the resulting figures when *puton* succeeds.

1) *Superposition on B/W type*: Assume that we superpose some unit on the B-type. The resultant figure is effective if and only if we superpose the straight-plate-unit, and it is valid for any type.

In contrast, it is impossible to place any unit on the W-type.

2) *Superposition of straight-plate-units*: Assume that we superpose the straight-plate-unit on the straight-plate-unit. The resultant figure is not always valid because its entire shape may not be a rectangle. The resultant figure is always effective.

3) *Superposition of the straight-plate-unit on the cross-plates-unit*: In this case, the resultant figure is not always valid and not always effective.

4) *Superposition of the cross-plates-unit on any type*: In this case, the resultant figure is always invalid in case of putting on the straight-plate-unit, but sometimes valid in case of putting on cross-plates-unit. It is always ineffective. However, the *puton* operation succeeds for several cases.

D. Success of extended puton operation

Here, we show the conditions under which the *puton* operation succeeds for any pair of units. In general, when the *puton* operation is performed on X and Y , WHITE should

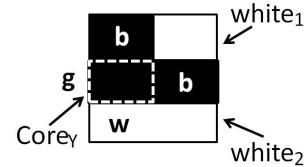


Figure 9. Representation of locations of white regions

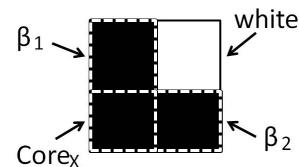


Figure 10. The regions to be hidden in L1-type

not be placed on X 's white region. When Y is a cross-plates-unit, we have to consider its white region located in the inclined orientation from $Core_Y$. The location of the white region is represented as the occurrence either of b in adjacent elements or of b and w in adjacent elements in the representation for Y . For example, a representation for a unit in Figure 9 is $\langle b, b, w, g \rangle$. The sequence b, b represents the location of $white_1$, the upper left of $Core_Y$, and the sequence b, w represents that of $white_2$, the lower part of the unit. Therefore, the condition on WHITE can be represented as (c5).

$$(c5) \quad \text{Let } \langle r_1, r_2, r_3, r_4 \rangle \text{ and } \langle r'_1, r'_2, r'_3, r'_4 \rangle \text{ be representations of } X \text{ and } Y, \text{ respectively. There exists some } i \ (1 \leq i \leq 4) \text{ such that } r_i = r'_{i+2} = g, \text{ where } r'_5 \text{ and } r'_6 \text{ are regarded as } r'_1 \text{ and } r'_2, \text{ respectively.}$$

Success of extended *puton* operation

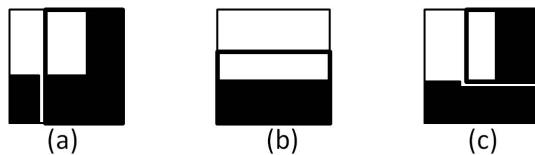
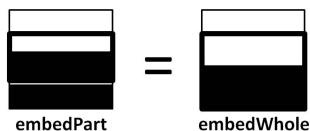
For any pair of units X and Y , if (c1) and (c5) are satisfied, the *puton* operation succeeds.

The *puton* operation is an operation of superposing core regions. We can consider another operation in the manner in which specified parts of both units are superposed, for example, superposing non-core regions. However, no manner of superposition other than *puton* operation will yield an effective solution. We will prove this property.

Theorem 7. *When we superpose the straight-plate-unit on the cross-plates-unit, only the *puton* operation will yield an effective solution.*

Proof:

Consider the *puton* operation that places a straight-plate-unit Y on an L1-type unit X shown in Figure 10. In this case, BLACK is divided into three regions: one core region $Core_X$ and two non-core regions β_1 and β_2 . Let $Core_Y$ be Y 's core region.

Figure 11. Three patterns of *embed* operationFigure 12. *embedWhole* corresponding to *embedPart*

One or two of the $Core_X, \beta_1, \beta_2$ should be hidden so that the resultant superposed figure is effective.

(i) Only one region is hidden.

If only $Core_X$ is hidden, β_1 and β_2 , which are disconnected, are visible. Therefore, the result is not effective. If only β_1 is hidden, $Core_X, \beta_2$ and $Core_Y$ are visible in the resultant figure. Considering the relative position of $Core_X, \beta_1$ and β_2 , it is impossible to make a rectangle by merging $Core_X, \beta_2$ and $Core_Y$ and to hide β_1 at the same time. Therefore, the result is not effective. Similarly, the result is not effective if only β_2 is hidden.

(ii) Two regions are hidden.

Because β_1 and β_2 are disconnected, they are not simultaneously hidden by a single unit. If both $Core_X$ and β_1 are hidden, β_2 and $Core_Y$ are visible. We must place Y 's regions onto both $Core_X$ and β_1 to hide them. Moreover, we must make a rectangle by merging β_2 and $Core_Y$. The only place where $Core_Y$ may be placed to satisfy both conditions is $Core_X$, and this placement is identical to the *puton* operation.

According to the above analysis, the resultant figure is not effective by any operation other than the *puton* operation.

Other cases can be similarly proved. \square

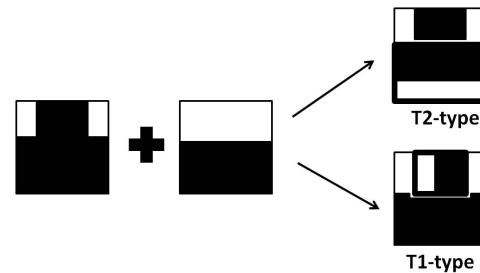
V. REASONING ABOUT SUPERPOSITION: EMBED

A. Superposition by *embedding*

We can consider another superposition operation of *embed*. This operation embeds the whole of one unit into the whole or a part of BLACK in the other unit. It is defined on a pair of types, while the *puton* operation is defined on a pair of representations for units.

Three patterns of embedding are possible, depending on the place of embedding.

- 1) Embed both into the core region and non-core region. For example, Figure 11(a) shows embedding of L1-type unit into L1-type unit.

Figure 13. Solution differences between *embedWhole* and *embedPart*

- 2) Embed only into the core region. For example, Figure 11(b) shows embedding of I1-type unit into I1-type unit. It is possible only when the background unit is straight-plate-unit.
- 3) Embed only into the non-core region. For example, Figure 11(c) shows embedding of I1-type unit into L1-type unit.

Remind the concept of a plate that is used in the construction of a valid unit. Two types of *embed* operation can be defined using this plate.

Definition 8. *Placement of the whole unit in its entirety on a plate of the other unit is called an *embedWhole* operation and placement on part of a plate of the other unit is called an *embedPart* operation.*

The first case and the second case in the above patterns are *embedWhole* operations, while the third case is an *embedPart* operation.

B. Result of superposition: *embed*

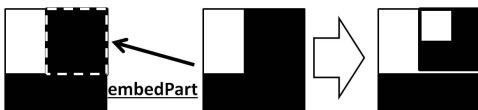
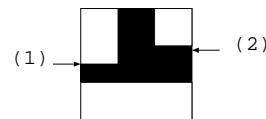
Next, we discuss the result of *embed* operation. The *embed* operation always succeeds in the sense that any region is not put on WHITE of the background unit, and the entire shape of the resultant figure is a rectangle. Therefore, we discuss only the validity and effectiveness of the resultant figure.

1) *Superposition on B/W type:* Assume that we superpose some unit on the B-type. The resultant figure is effective if and only if we superpose the straight-plate-unit, and it is valid for any type.

In contrast, it is impossible to place any unit on the W-type.

2) *Superposition of straight-plate-units:* Assume that we superpose the straight-plate-unit on the straight-plate-unit. The resultant figure obtained by the *embed* operation is not always valid because the white region may not be convex. The resultant figure is always effective.

Theorem 9. *If the result of *embedPart* operation on a pair of straight-plate-units is valid and effective, then*

Figure 14. Ineffectiveness of *embedPart* for a pair of cross-plates-unitsFigure 15. U^* : Invalid example

an *embedWhole* operation exists that generates the same result.

Proof.

Assume that the result of *embedPart* operation on a pair of straight-plate-units is valid and effective, shown in Figure 12, for example. If we extend the BLACK of the foreground unit to fill the core region of the background unit, this corresponds to the *embedWhole* operation, which generates the same result. \square

It means that two figures in Figure 12 are regarded as qualitatively equivalent, and this is a characteristic of qualitative reasoning.

3) *Superposition of the straight-plate-unit on the cross-plates-unit*: In this case, the resultant figure obtained is not always valid and not always effective. *embedWhole* and *embedPart* may generate different solutions for the same pair. For example, if an I1-type unit is embedded into a T1-type unit, T2-type is generated by *embedWhole*, while T1-type is generated by *embedPart* (Figure 13).

4) *Superposition of the cross-plates-unit on any type*: In this case, the resultant figure is always ineffective but can yield valid figures in some cases (See Table I).

Moreover, the following property holds.

Theorem 10. *If the result of embed operation on a pair of cross-plates-units is valid, then it is an embedWhole operation.*

Proof.

Assume that the result of *embedPart* is valid. In this case, BLACK portions of at least one plate of the background unit in the resulting figure are visible (Figure 14). These parts are the ones in which no unit is embedded. In contrast, BLACK of the foreground unit is visible and is not rectangular. The union of these portions of BLACK cannot make a shape of BLACK for any unit. Therefore, *embedPart* never generates a valid solution. \square

5) *Valid solutions*: Table I shows a pair of unit types where *embed* operation generates a valid solution. In this table, rows show the unit in the foreground, and columns show the unit in the background. Only the solutions that differ from those generated by *puton* operations are shown. U means there is no solution. The case of U^* appears to be successful at first glance, but there is actually no solution. For example, Figure 15 shows the resultant figure obtained by the operation of *embed* for L2 on T1. It is not valid because it is impossible to align line (1) and line (2).

Unlike the *puton* operation, we cannot currently formalize rules for selecting the proper position for embedding or orientating units to obtain valid solutions. We can only say that generally, we place WHITE of a pair of superposing units on the adjacent position.

C. Comparison with *puton* operation

Table II compares validity and effectiveness between *puton* and *embed* operations. In the table, s and c indicate straight-plate-unit and cross-plates-unit, respectively.

Theorem 11. (1) *If there is a valid solution for the puton operation, then there is a valid solution by the embedWhole operation that generates the same solution.*

(2) *Even if there is no valid solution for the puton operation, there may be a valid solution by the embedWhole operation.*

Proof.

(1) We cannot obtain a valid solution in the case applying the *puton* operation for cross-plates-unit on straight-plate-unit. Therefore, we consider the remaining three cases.

(i) superposing cross-plates-unit on cross-plates-unit

As both core regions are superposed, a plate of the foreground unit is put on a plate of the background unit. Let A be a visible part of a background unit and B be its invisible part. Moreover, let A' be a part of the foreground unit that is placed on the background unit and B' be its remaining part. Then, A' is a foreground of B by *puton* operation. If we extend B so that it is a background of both A' and B' , it is a solution of *embed* operation. Since B corresponds to a single plate, its extention is qualitatively equivalent to the original one (Figure 16).

(ii) superposing straight-plate-unit on straight-plate-unit

This is proved in a similar way to the case (i).

(iii) superposing straight-plate-unit on cross-plates-unit

It is trivial due to the validity of the resultant figure.

(2) Only the solutions that differ from those generated by *puton* operations are shown in Table I. \square

embed is an operation that is as essential as *puton*. When we superpose multiple units using both operations, we can sometimes obtain solutions that are not obtained only by a single operation. We can illustrate this using example of superposition of four units.

Consider superposition of four units X, Y, Z and W shown in Figure 17. The representations for X, Y, Z and W are $\langle b, b, g, b \rangle$, $\langle b, b, g, b \rangle$, $\langle w, g, w, g \rangle$, and $\langle w, w, w, w \rangle$,

fg \ bk	I1	I2	L1	L2	T1	T2	PLUS
I1	I1	I2	T1 L1 L2	T2 L2	T1 T2	T2	U*
I2	I2	I2	T1	T2	T1	T2	U
L1	L2	U	L1	L2 T2	T1	T2	U*
L2	L2	U	L2	L2	U*	U*	U
T1	T2	U	T1	T2	T1	T2	PLUS
T2	T2	U	U	U	T2	T2	U
PLUS	U	U	U	U	PLUS	U	PLUS

Table I
VALID SOLUTIONS FOR THE *embed* OPERATION

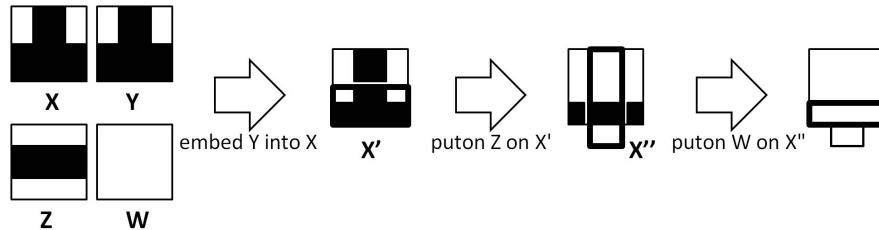


Figure 17. Solution by using *puton* and *embed* operations together

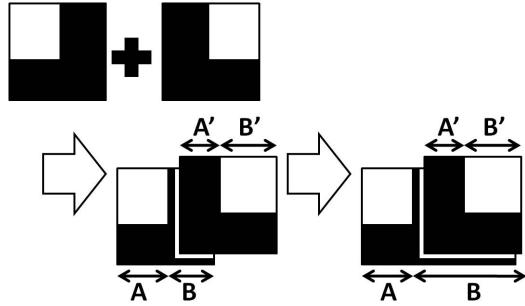


Figure 16. Superposing cross-plates-unit on cross-plates-unit

<i>puton</i>		<i>embed</i>	
s on s	some	always	some
s on c	some	some	some
c on c	some	never	some
c on s	never	never	some

Table II
COMPARISON BETWEEN *puton* AND *embed*

respectively. First, an effective solution for X, Y and Z should be generated. *puton* operation succeeds only for X and Z , that is, $\text{puton}(X, Z) = \langle w, g, w, g \rangle$, but the resultant figure is not valid. Therefore, we cannot continue the operation. In contrast, *embed* X into Y generates a valid solution X' . Next, put Z on this result X' using the *puton* operation. This yields an effective solution X'' . Finally, by putting W on this result X'' using the *puton* operation we obtain the solution for superposing the four units.

Let Ω be a finite set of valid units that does not include a W -type unit, where $|\Omega| \geq 2$, and w is a W -type unit.

- (1) Extract an arbitrary pair of X and Y from Ω .
- (2) If superposing Y on X generates a valid and effective solution,
let Z be the resulting figure.
Otherwise, go back (1) to find another pair.
- (3) If $|\Omega| = 2$,
if Z is effective, then Z is a solution.
else go back (1) to find another pair.
else continue.
- (4) Set $\Omega = \Omega - \{X, Y\} \cup \{Z\}$, and go to (1).

If a solution is generated, then the superposition of $\Omega \cup \{w\}$ succeeds.
If it fails in all cases, then there is no solution.

Figure 18. Algorithm for superposing multiple units

VI. RECTANGLE REASONING SYSTEM

A. Algorithm for multiple unit superposition

We explain an algorithm for superposing multiple units. Here, superposition means either *puton* or *embed* operation. Selecting a unit from a given set of valid units, and perform superposition operation repeatedly to find a solution. The algorithm is shown in Figure 18.

B. Reasoning system

We implemented this algorithm using Prolog to code the main reasoning part and Java for the interface part.

We explain the behavior of the system.

1) Initial state

When the system is invoked, a basic frame is displayed that shows nine types of units (Figure 19).

2) Selecting the set of units

First, determine the first unit for superposing in the following manner: select the unit type by pushing the "select" button from "action" on the menu bar. Determine its orientation by pushing the "rotation" button. Repeat this procedure until n units are determined. Multiple units of the same type may be selected. The selected units are shown on the lower part of the frame (Figure 20).

3) Superposition

Next, judge if superposition succeeds and generate the solution if one is available. Find the superposing manner by pushing the "start" button from "file" on the menu bar. This opens a new window displaying the result. If superposition succeeds, the order of superposition and the positions of each unit are displayed (Figure 21). Otherwise, the window displays "No solution." If more than one solution is possible, only the first one found is shown.

VII. CONCLUSION AND FUTURE WORK

A. Conclusions

We have discussed superposition of a pair of units and investigated the conditions that satisfy the result where all white regions are visible while all black regions are hidden in the resultant figure when visibility is specified by a user.

- A pair of straight-plate-units always produces an effective solution either by the *puton* operation or by the *embed* operation.
- The straight-plate-unit on cross-plates-unit can produce an effective solution in some cases either by the *puton* operation or by the *embed* operation. If a solution generated by the *puton* operation is valid, then it is also generated by the *embed* operation.
- The cross-plates-unit on any type can produce no effective solution.

As for the last case, we have shown which pairs can generate valid solutions.

We also presented an algorithm for superposing a set of units and implemented this system.

This work is the first study to focus on object placement with superposition and demonstrates a new application of QSR.

B. Future Works

We admit only units constructed using two specific plates. As a result, we have constraints both on BLACK and WHITE of a unit: BLACK should be one connected and all white regions are rectangles.



Figure 22. A unit constructed using three plates



Figure 23. Allowing a non-rectangular white region

Assume that we admit a unit obtained by packing three plates (Figure 22). The above constraint on BLACK still exists. In this case, there are two core regions.

Next, consider that we weaken the constraint on WHITE. Assume that we can admit a white region that is not rectangular (Figure 23). In this case, a white region exists at the position over the black region viewed from the core region, whereas in the current definition, a white region can exist either on the adjacent or inclined orientation of the core region.

Currently, we can represent each unit uniquely by a representation using four directions of the core region. However, if we weaken the constraints and extend the target objects, we must change this representation, as the above consideration shows. This process is not straightforward, but we hope to weaken these constraints in future.

REFERENCES

- [1] G. Birgin, R. D. Lobato, and R. Morabito, "An effective recursive partitioning approach for the packing of identical rectangles in a rectangle," *Journal of the Operational Research Society*, vol. 61, pp. 306-320, 2010.
- [2] A. S. Lapaugh, "Layout algorithm for VLSI design," *ACM Computing Surveys*, vol. 28, no. 1, pp. 59-61, 1996.
- [3] H. Freeman, "Computer name placement," in *Geographical Information Systems I*, D. J. Maguire, M. F. Goodchild, and D. W. Rhind, Eds. John Wiley, 1991, pp. 449-460.
- [4] J. Li, C. Plaisant, and B. Shneiderman, "Data object and label placement for information abundant visualizations," in *Proceedings of the Workshop of New Paradigms Information Visualization and Manipulation (NPIV98)*, 1998, pp. 41-48.
- [5] M. Aliello, I. E. Pratt-Hartmann, and J. F. A. K. Van Benthem, Eds., *Handbook of Spatial Logics*. Springer-Verlag, 2007.
- [6] A. Cohn and S. Hazarika, "Qualitative spatial representation and reasoning: an overview," *Fundamental Informaticae*, vol. 46, no. 1, pp. 1-29, 2001.
- [7] A. Cohn and J. Renz, "Qualitative spatial representation and reasoning," *Handbook of Knowledge Representation*, Chapt. 13, pp. 551-596, F. van Harmelen, V. Lifschitz, and B. Porter, Eds., Elsevier, 2008.
- [8] M. Egenhofer and R. Franzosa, "On the equivalence of topological relations," *International Journal of Geographical Information Systems*, vol. 9, no. 2, pp. 133-152, 1995.
- [9] S. Kumokawa and K. Takahashi, "Rectangle reasoning: a qualitative spatial reasoning with superposition," in *Proceedings of 23rd Florida Artificial Intelligence Research Society Conference (FLAIRS23)*, 2010, pp. 150-151.

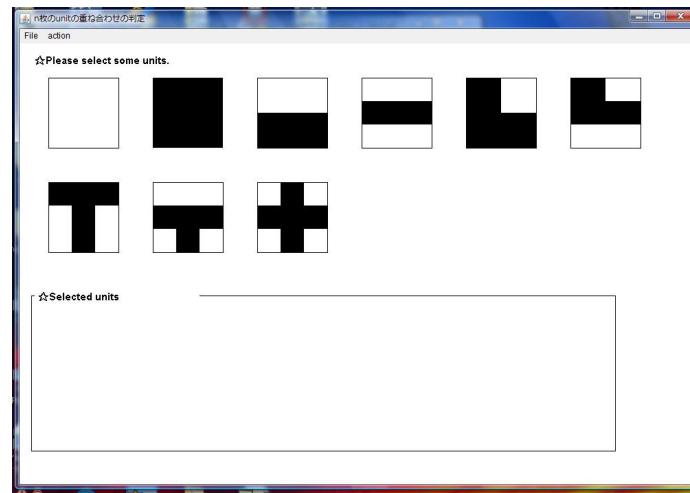


Figure 19. Screenshot of the system: 1. Initial state

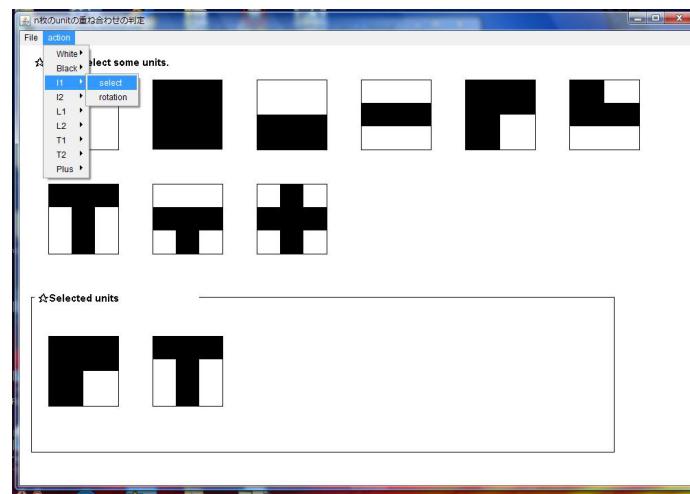


Figure 20. Screenshot of the system: 2. Selecting the set of units

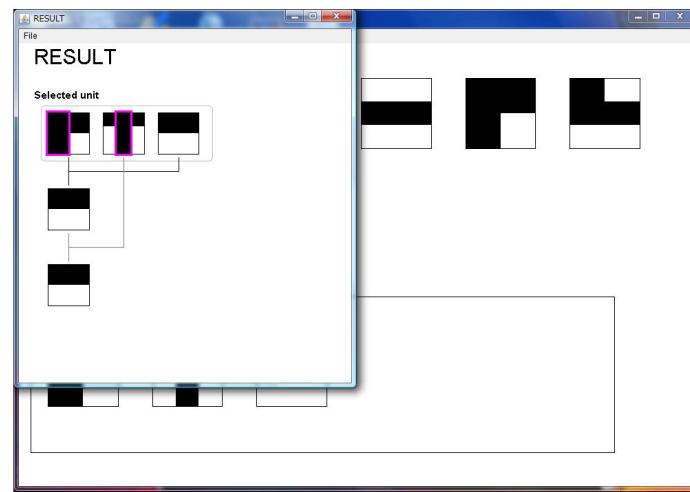


Figure 21. Screenshot of the system: 3. Superposition

- [10] T. Konishi, and K. Takahashi, "Symbolic Representation and Reasoning for Rectangles with Superposition," The Third International Conference on Advances in Databases, Knowledge, and Data Applications (DBKDA 2011), pp. 71-76, January, 2011.
- [11] S. Wang and D. Liu, "Qualitative spatial relation database for semantic web," in *First Asian Semantic Web Conference (ASWC)*, 2006, pp. 387-399.
- [12] M. Santos and L. Amaral, "Geo-spatial data mining in the analysis of a demographic database," *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, vol. 9, no. 5, pp. 374-384, 2005.

Efficient Non-Sequential Access and More Ordering Choices in a Search Tree

Lubomir Stanchev

Computer Science Department

Indiana University - Purdue University Fort Wayne

Fort Wayne, IN, USA

stanchel@ipfw.edu

Abstract—A traditional search tree allows for efficient sequential access to the elements of the tree. In addition, a search tree allows for efficient insertion of new elements and efficient deletion of existing elements. In this article we show how to extend the capabilities of a search tree by presenting an algorithm for efficient access to predefined subsets of the indexed elements. This is achieved by marking some of the elements of the search tree with marker bits. In addition, our algorithm allows us to efficiently retrieve the indexed elements in either ascending or descending direction relative to each of the ordering attributes.

Index Terms—marker bits; search trees; ordering directions; data structures

I. INTRODUCTION

The paper extends a conference paper on the topic of efficient access to non-sequential elements of a search tree ([5]). The major contribution of this article is the introduction of an algorithm that extends the ordering choices for the elements that are returned. As a minor contribution, we show that retrieving multiple elements from a search tree with marker bits takes time that is proportional to the size of the tree.

A balanced search trees, such as an AVL tree ([1]), an AA tree ([2]), or a B^+ tree ([3]), allows efficient retrieval of elements that are consecutive relative to an in-order traversal of the tree. However, there is no obvious way to efficiently retrieve the elements that belong to a predefined subset of the stored elements if they are not sequential in the search tree. Similarly, there is no obvious way of retrieving the elements in an order that is different from the tree order (or the reverse of the tree order). For example, consider a database that stores information about company employees. A search tree may store information about the employees ordered first by age ascending and then by name ascending. This search tree can be used to retrieve all the employees sorted by age, but the search tree does not efficiently support the request of retrieving all rich employees (e.g., making more than \$100,000 per year) sorted by age. In this paper, we will show how the example search tree can be extended with marker bits so that both requests can be efficiently supported. In addition, we will show how the search tree can be used to efficiently retrieve the elements in a different order, for example, ordered by age descending (rather than ascending) and then by name ascending.

The techniques that are proposed in this paper will increase the set of requests that can be efficiently supported by a search tree. This means that fewer search trees will need to be built. This approach will not only save space, but will also improve update performance. For example, [6] shows how our approach can be applied to perform index merging. Specifically, indices on the same elements that have different orderings can be merged when the orderings are on the same attributes. Similarly, indices with orderings on the same attributes that contain common elements can be merged together.

Naïve solutions to the problem of efficiently accessing a non-sequential subset of the elements that are indexed fails. For example, it is not enough to mark all the nodes of the search tree that contain data elements that belong to the subset. This approach will not allow us to prune out subtrees because it can be the case that the parent node does not belong to an interesting subset, but some of the descendant nodes do belong. Similarly, efficiently accessing the elements of a search tree where the ascending and descending direction of the attributes is changed is not trivial because this can require both forward and backward scanning.

To the best of our knowledge, detailed explanation of how marker bits work have not been previously published. Our previous work [6] briefly introduces the concept of marker bits, but it does not explain how marker bits can be maintained after insertion, deletion, or update. Other existing approaches handle requests on different subsets of the search tree elements by exhaustive search or by creating additional search trees. However, the second approach leads to not only unnecessary duplication of data, but also slower updates to multiple copies of the same data. Similarly, to the best of our knowledge, no previous research addresses the problem of efficiently retrieving the elements of a search tree in order that is different from the search order or its reverse.

Given a subset of the search elements S , our approach to efficiently retrieve these elements marks every node in the tree that contains an element of S or that has a descendant that contains an element of S . These additional marker bits will only slightly increase the size of the search tree (with one bit per tree node), but will allow efficient logarithmic execution of requests that ask for the elements of S in the tree order. It will take time proportional to the size of the tree to retrieve all the elements of S .

The algorithm that returns the elements of a search tree in an order that changes the ascending and descending direction of the attributes repeatedly calls the `next` method. The method tries to find the “next” element that has the same value for all but the last attribute as the current node, where `next` is defined relative to the direction of the last attribute. If such an element does not exist, then the method tries to find the next element that has the same value for all but the last two attributes and so on, where the `next` method is recursive.

In what follows, Section II presents core definitions, Section III describes how to efficiently perform different operations on a search tree with marker bits, and Section IV contains the conclusion and directions for future research.

II. DEFINITIONS

Definition 1 (MB-tree): An MB-tree has the following syntax: $\langle\langle S_1, \dots, S_s \rangle, S, O \rangle$, where S and $\{S_i\}_{i=1}^s$ are sets over the same domain Δ , $S_i \subseteq S$ for $i \in [1..s]$, and O is an ordering over Δ . This represents a balanced search tree of the elements of S (every node of the tree stores a single element of S), where the in-order traversal of the tree produces the elements according to the order O . In addition, every node of the tree contains s marker bits and the i^{th} marker bit is set exactly when the node or one of its descendants stores an element that belongs to S_i - we will refer to this property as the *marker bit property*.

The above definition can be extended to allow an MB-tree to have multiple data values in a node, as is the case for a B Tree. However, this is area for future research.

Going back to our motivating example, consider the MB-tree $\langle\langle RICH_EMPS \rangle, EMPS, \langle age \text{ asc}, name \text{ asc} \rangle \rangle$. This represents a search tree of the employees, where the ordering is first relative to the attribute *age* in ascending order. If two employees have the same age, then they are ordered relative to their name in ascending order. The *RICH_EMPS* set consists of the employees that make more than \$100,000 per year. Figure 1 shows an example instance of this MB-tree. Each node of the tree contains the name of the employee followed by their age and salary.

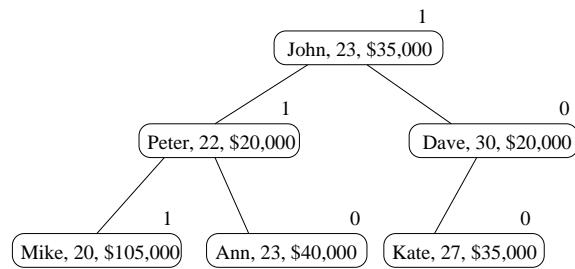


Fig. 1. Example of an MB-tree

Each node in the MB-tree contains the name of the employee, their age, and their salary. Note that Ann and John are the same age. However, Ann comes before John in the search order because “Ann” is lexicographically before “John”.

TABLE I
INTERFACE OF A NODE

(operation)	(return value)
left ()	left child
right ()	right child
parent ()	parent node
data ()	stored data
m[i]	the i marker bit ($1 \leq i \leq s$)

Above each node, the value of the marker bit is denoted, where the bit is set exactly when the node or one of its descendants contains a rich employee. As the figure suggests, the subtree with root node that contains the name Dave can be pruned out when searching for rich employees because the marker bit of the root node is not set. We will show that this MB-tree can be used to efficiently find not only all employees sorted by age, but also all rich employees sorted by age.

The tree can also be used to efficiently find employee (or rich employees) ordered by, for example, age descending and then name ascending. The query that is asking for all employees will return the employees in order: Dave, Kate, Ann, John, Peter, and Mike, while the second query will return only Mike (the only rich employee). Note that throughout this paper *efficient* refers to logarithmic time relative to the size of the search tree.

III. OPERATIONS ON AN MB-TREE

Although an MB-tree does not need to be binary, in the following section we will consider only binary trees. The presented algorithms can be extended to non-binary trees and this is area for future research. We will assume that every node of the search tree supports the methods of the interface that is shown in Table I in constant time, where $\{S_i\}_{i=1}^s$ are the marker bit sets.

Next, we describe how the algorithms for tree update and search can be extended in the presence of marker bits. Note that supporting more ordering choices only affects the search algorithm.

A. Element Insertion

After an algorithm has inserted a leaf node n , it should call the `insert_fix` method from Algorithm 1 to update the marker bits in the tree.

Algorithm 1 `insert_fix` (Node n)

```

1: for  $i \leftarrow 1$  to  $s$  do
2:   if  $n.\text{data}() \in S_i$  then
3:      $n.m[i] \leftarrow 1$ 
4:   else
5:      $n.m[i] \leftarrow 0$ 
6:   end if
7: end for
8: insert_parent_fix ( $n.\text{parent}()$ ,  $n.m$ )
  
```

Lines 1-7 of the code set the marker bits for the new node. The call to the recursive procedure `insert_parent_fix`

fixes the marker bits of the ancestors of the inserted node, where the later procedure is presented in Algorithm 2.

Algorithm 2 insert_parent_fix(Node n , Bit[] m)

```

1: if  $n = \text{null}$  then
2:   return
3: end if
4:  $\text{changed} \leftarrow \text{false}$ 
5: for  $i \leftarrow 1$  to  $s$  do
6:   if  $m[i] = 1$  and  $n.m[i] = 0$  then
7:      $n.m[i] \leftarrow 1$ 
8:      $\text{changed} \leftarrow \text{true}$ 
9:   end if
10: end for
11: if  $\text{changed}$  then
12:   insert_parent_fix( $n.\text{parent}()$ ,  $n.m$ )
13: end if

```

We claim that the resulting tree satisfies the marker bit property. In particular, note that only the marker bits of the inserted node and its ancestors can be potentially affected by the insertion. Lines 1-7 of the `insert_fix` method update the marker bits of the node that is inserted. If the i^{th} marker bit of the node is set, then we check the i^{th} marker bit of its parent node (Lines 6 of the `insert_parent_fix` method). If the i^{th} marker bit of the parent is set, then the i^{th} marker bit of all ancestors will be set because of the marker bit property and nothing more needs to be done for the i^{th} marker bit. Conversely, if the i^{th} marker bit of the parent is not set, then we need to set it and then check the i^{th} marker bit of the parent of the parent node and so on. This is done by Line 7 and the recursive call at Line 12, respectively. The variable `changed` is used to record whether any of the marker bits of the current node have been modified. If the variable is equal to `true`, then the marker bits of the ancestor nodes will not need to be updated. Therefore, the marker bits of the inserted node and its ancestors are updated correctly and the marker bit property is preserved for the updated search tree.

B. Deleting a Node with Less than Two Children

Deleting a node with two children from a binary tree cannot be performed by just connecting the parent of the deleted node to the children of the deleted node because the parent node may end up with three children. Therefore, we will consider two cases: when the deleted node has less than two non-null children and when the deleted node has two non-null children. The first case is explained next, while the second case is explained in Section III-D.

An implementation of Algorithm 3 should be called before a node n with less than two non-null children is deleted. In the algorithm, $n.\text{child}()$ is used to denote the non-null child of n and $m[i]$ is set when the i^{th} marker bits of the ancestor nodes need to be checked. The algorithm for the method `delete_parent_fix` that updates the marker bits of n 's ancestors in the search tree is shown in Algorithm 4.

Algorithm 3 delete_fix_simple(Node n)

```

1: for  $i \leftarrow 1$  to  $s$  do
2:   if  $n.\text{data}() \in S_i$  and ( $n$  is leaf node or
    $n.\text{child}().m[i] = 0$ ) then
3:      $m[i] \leftarrow 1$ 
4:   else
5:      $m[i] \leftarrow 0$ 
6:   end if
7: end for
8: delete_parent_fix( $n.\text{parent}()$ ,  $m$ )

```

Algorithm 4 delete_parent_fix(Node n , Bit[] m)

```

1: if  $n = \text{null}$  then
2:   return
3: end if
4:  $\text{changed} \leftarrow \text{false}$ 
5: for  $i \leftarrow 1$  to  $s$  do
6:   if  $m[i] = 1$  and  $n.\text{data}() \notin S_i$  and ( $n$  has no other
      child or  $n.\text{other\_child}().m[i] = 0$ ) then
7:      $n.m[i] \leftarrow 0$ 
8:      $\text{changed} \leftarrow \text{true}$ ;
9:   end if
10: end for
11: if  $\text{changed}$  then
12:   delete_parent_fix( $n.\text{parent}()$ ,  $m$ )
13: end if

```

Note that we have used $n.\text{other_child}$ to denote the child node of n that is not on the path to the deleted node. We claim that the deletion algorithm preserves the marker bit property. In particular, note that only the ancestors of the deleted node can be affected. If $m[i] = 1$ (Line 6 of the `delete_parent_fix` method), then we check whether the data in the node belongs to S_i and whether the i^{th} marker bit of the other child node is set. If both conditions are false, then the only reason the i^{th} marker bit of n is set is because the data in the deleted node belonged to S_i and now this marker bit needs to be reset (Line 7) and the ancestors of n need to be recursively checked (Line 12). Conversely, if one of the conditions is true or $m[i] = 0$, then the i^{th} marker bit of n and its ancestors will not be affected by the node deletion. Therefore, the marker bits of the ancestors of the deleted node are updated correctly and the marker bit property holds for the updated search tree.

C. Element Update

Algorithm 5 should be executed after the data in a node n is modified, where v is the old data value of n . The pseudo-code updates the marker bits of the node n and then calls the `update_parent_fix` method, which is presented in Algorithm 6.

Note that we have used $n.\text{other_child}()$ to denote the child node of n that is not on the path to the updated node. The method `update_fix` preserves the marker bit

Algorithm 5 update_fix(Node n , Value v)

```

1:  $old \leftarrow n$ 
2: for  $i = 1$  to  $s$  do
3:   if  $n.data() \in S_i$  or ( $n.left() \neq null$  and
     $n.left().m[i] = 1$ ) or ( $n.right() \neq null$  and
     $n.right().m[i] = 1$ ) then
4:      $n.m[i] \leftarrow 1$ 
5:   else
6:      $n.m[i] = 0$ 
7:   end if
8:   if  $n.m[i] = 1$  and  $old.m[i] = 0$  then
9:      $m[i] \leftarrow "insert"$ 
10:   else if  $n.m[i] = 0$  and  $old.m[i] = 1$  then
11:      $m[i] \leftarrow "delete"$ 
12:   else
13:      $m[i] \leftarrow "no change"$ 
14:   end if
15: end for
16: update_parent_fix( $n.parent()$ ,  $m$ )

```

Algorithm 6 update_parent_fix(Node n , Value[] m)

```

1: if  $n = null$  then
2:   return
3: end if
4:  $changed \leftarrow false$ 
5: for  $i = 1$  to  $s$  do
6:   if  $m[i] = "insert"$  and  $n.m[i] = 0$  then
7:      $n.m[i] \leftarrow 1$ 
8:      $changed \leftarrow true$ 
9:   end if
10:  if  $m[i] = "delete"$  and  $n.data() \notin S_i$ 
    and ( $n.other\_child() = null$  or
     $n.other\_child().m[i] = 0$ ) then
11:     $n.m[i] \leftarrow 0$ 
12:     $changed \leftarrow true$ 
13:  end if
14: end for
15: if  $changed$  then
16:   update_parent_fix( $n.parent()$ ,  $m$ )
17: end if

```

property because it is a combination of the `insert_fix` and `delete_fix_simple` methods. In particular, $m[i]$ in the method `update_fix` is set to `insert` when the i^{th} marker bit of the updated node was changed from 0 to 1 and to `delete` when this marker bit was updated from 1 to 0. The first case is equivalent to a node with the i^{th} marker bit set being inserted, while the second case is equivalent to a node with the i^{th} marker bit set being deleted.

D. Deleting a Node with Two Children

As it is usually the case ([4]), we assume that the deletion of a node n_1 with two non-null children is handled by first deleting the node after n_1 relative to the tree order, which

we will denote as n_2 , followed by changing the data value of n_1 to that of n_2 . The pseudo-code in Algorithm 7, which implementation should be called after a node is deleted from the tree, shows how the marker bits can be updated, where initially $n = n_1$, p is the parent of n_2 , v is the value of the data that was stored in n_2 , and $m[i] = 1$ exactly when $n_2.m[i] = 1$ and $n'.m[i] = 0$ for all descendants n' of n_2 .

Algorithm 7 delete_fix_two_children(n, p, v, m)

```

1: if  $p = n$  then
2:   update_fix( $n, v$ )
3: end if
4:  $changed \leftarrow false$ 
5: for  $i=1$  to  $s$  do
6:   if  $m[i] = 1$  and  $p.data() \notin S_i$  and ( $p$  has no other
    child or  $p.other\_child().m[i] = 0$ ) then
7:      $p.m[i] \leftarrow 0$ 
8:      $changed \leftarrow true$ 
9:   end if
10: end for
11: if  $changed$  then
12:   delete_fix_two_children( $n, p.parent()$ ,
     $v, m$ )
13: else
14:   update_fix( $n, v$ )
15: end if

```

In the above code “ p has no other child” refers to the condition that p has no other child than the child that it is on the path to the deleted node n_2 . Similarly, `p.other_child()` is used to denote the child of p that is not on the path to the deleted node n_2 . Note that the above algorithm changes the nodes on the path from n_2 to n_1 using the deletion algorithm from the method `delete_parent_fix` and the nodes on the path from n_1 to the root of the tree using the update algorithm from the method `update_fix` and is therefore correct.

E. Tree Rotation

Most balancing algorithms (e.g., the ones for AVL, red-black, or AA trees) perform a sequence of left and/or right rotations whenever the tree is not balanced as the result of some operation. Here, we will describe how a right rotation can be performed, where the code for a left rotation is symmetric. The pseudo-code in Algorithm 8 should be called with a parent node n_2 and a right child node n_1 after the rotation around the two nodes was performed. The pseudo-code only fixes the marker bits of n_1 and n_2 . The descendants of all other nodes will not change and therefore their marker bits do not need to be updated.

F. Time Analysis for the Modification Methods

Obviously, the pseudo-code for the rotation takes constant time. The other methods for updating marker bits visit the updated node and possibly some of its ancestors and perform constant number of work on each node and therefore take order logarithmic time relative to the number of nodes in the tree.

Algorithm 8 `rotate_right_fix(n_1, n_2)`

```

1: for  $i \leftarrow 1$  to  $s$  do
2:   if  $n_1.\text{data}() \in S_i$  or ( $n_1$  has left child and
    $n_1.\text{left}().m[i] = 1$ ) then
3:      $n_1.m[i] \leftarrow 1$ 
4:   end if
5:   if  $n_2.\text{data}() \in S_i$  or ( $n_2$  has left child and
    $n_2.\text{left}().m[i] = 1$ ) or ( $n_2$  has right child and
    $n_2.\text{right}().m[i] = 1$ ) then
6:      $n_2.m[i] \leftarrow 1$ 
7:   end if
8: end for

```

Therefore, the extra overhead of maintaining the marker bits will not change the asymptotic complexity of the modification operations.

G. Search

Let us go back to our motivating example from Figure 1. Our desire is to efficiently retrieve all rich employees in the tree order. This can be done by repeatedly calling the implementation of the `next` method from Algorithm 9. The terminating condition is when the method returns `null`. The algorithm finds the first node after n , relative to the tree order, that has data that belongs to the set S_i , where d is initially set to `false`.

Algorithm 9 `next(n, i, d)`

```

1: if  $n.\text{data}() \in S_i$  and  $d$  then
2:   return  $n$ 
3: end if
4: if  $n.\text{left}()$  is not the last node visited and  $n.\text{left}() \neq$ 
   null and  $n.\text{left}().m[i] = 1$  and  $d$  then
5:   return next( $n.\text{left}(), i, \text{true}$ )
6: end if
7: if  $n.\text{right}()$  is not the last node visited and
    $n.\text{right}() \neq \text{null}$  and  $n.\text{right}().m[i] = 1$  then
8:   return next( $n.\text{right}(), i, \text{true}$ )
9: end if
10: if  $n.\text{parent}() = \text{null}$  then
11:   return null
12: end if
13: return next( $n.\text{parent}(), i, d$ )

```

The algorithm first checks if the data in the current node is in S_i and d is true. Note that d becomes true when n is a node that is after the initial node in the search tree. When both conditions are true, we have found the resulting node and we just need to return it. Next, we check the left child node. If we did not just visit it and its i^{th} bit is marked and it is after the start node relative to the in-order tree traversal order, then the subtree with root this node will contain a node with data in S_i that will be the resulting node. Next, we check if the right child has its i^{th} bit marked. This condition and the condition that we have not visited it before guarantees that this subtree

will contain the resulting node. Finally, if neither of the child subtrees contain the node we are looking for, we start checking the ancestor nodes in order until we find an ancestor that has a right child node that we have not visited and its i^{th} marker bit for this child is set. We then visit this subtree because we are guaranteed that it will contain the resulting node. Therefore, the algorithm finds the first node after n that has data that is in S_i . Since, in the worst case, we go up a path in the search tree and then down a path in the search tree, our worst-case asymptotic complexity for finding the next node with data in S_i is logarithmic relative to the size of the tree, which is the same as the asymptotic complexity of the traditional method for finding a next element in a balanced search tree.

Next, we will consider a method that finds all the elements in the search tree without using the `next` method and we will show that this method runs in time that is proportional to the size of the tree. Note that, in the worst case all nodes belong to the query result and therefore we cannot do any better. The algorithm is presented in Algorithm 10. In the method, n is initially the root node of the search tree. Since the method visits every node once, it runs in time that is proportional to the size of the tree. Note that the nodes that are visited by calling the `next` method multiple times are the same as the nodes that are visited by calling the `find_all` method. In both cases, the nodes in the tree are visited relative a in-order traversal of the tree, where subtrees that have root nodes that are unmarked are pruned out.

Algorithm 10 `find_all(n, i)`

```

1: result  $\leftarrow \emptyset$ 
2: if  $n.\text{left}() \neq \text{null}$  and  $n.\text{left}().m[i] = 1$  then
3:   result  $\leftarrow \text{result} \cup \text{find\_all}(n.\text{left}(), i)$ 
4: end if
5: if  $n.m[i] = 1$  then
6:   result  $\leftarrow \text{result} \cup \{n\}$ 
7: end if
8: if  $n.\text{right}() \neq \text{null}$  and  $n.\text{right}().m[i] = 1$  then
9:   result  $\leftarrow \text{result} \cup \text{find\_all}(n.\text{right}(), i)$ 
10: end if
11: return result

```

Next, we will describe a search algorithm that can be used to efficiently retrieve the elements of the search tree in an order that is different from the search order. For starters, we present the method `previous` that returns the previous element that belongs to the set S_i . The method is presented in Algorithm 11, where d is initially set to `false`. Note that the method `previous` is analogous to the method `next`. To only difference is that it searches for an element to the left (rather than to the right) of the current element.

Next, we present a method `search` that is also need in order to retrieve the elements of the search tree in an order that is different from the search tree order. The method has the following properties, where we assume that the search tree contains elements with attributes $\{A_i\}_{i=1}^a$ and that the ordering of the tree is $\langle A_1 \text{ asc}, \dots, A_a \text{ asc} \rangle$.

Algorithm 11 previous(n, i, d)

```

1: if  $n.\text{data}() \in S_i$  and  $d$  then
2:   return  $n$ 
3: end if
4: if  $n.\text{right}()$  is not the last node visited and
    $n.\text{right}() \neq \text{null}$  and  $n.\text{right}().m[i] = 1$  and  $d$ 
   then
5:   return previous( $n.\text{right}(), i, \text{true}$ )
6: end if
7: if  $n.\text{left}()$  is not the last node visited and  $n.\text{left}() \neq$ 
   null and  $n.\text{left}().m[i] = 1$  then
8:   return previous( $n.\text{left}(), i, \text{true}$ )
9: end if
10: if  $n.\text{parent}() = \text{null}$  then
11:   return  $\text{null}$ 
12: end if
13: return previous( $n.\text{parent}(), i, d$ )

```

search($A_1, P_1, \dots, A_l, P_l, \text{dir}, r, i$):

- **pre-conditions:** $l \leq a$ and $r \in [l-1, l]$.
- **return value:** If $\text{dir} = \text{asc}$ ($\text{dir} = \text{desc}$), then this method returns the the first (last) node n in S_i , relative to the tree order, for which:

- 1) $\bigwedge_{i=1}^r (n.\text{data}.A_i = P_i)$ and
- 2) if $r < l$, then $n.\text{data}.A_l > P_l$ when $\text{dir} = \text{asc}$ and
 $n.\text{data}.A_l < P_l$ when $\text{dir} = \text{desc}$.

The method returns null when such a node does not exist.

The method search is used to search for the node that has the same value for some of the attributes as the current node, which allows us to go forward and backwards in the tree and return the elements in the desired order. The pseudo-code for the method when $r = l$ is presented in Algorithm 12. Note that we have added a node n as a parameter, which is initially the root of the search tree. The code when $\text{dir} = \text{asc}$ and $\text{dir} = \text{desc}$ are symmetric. The expression $\langle P_1, \dots, P_l \rangle \prec n$ returns true when $P_j \leq n.\text{data}().A_j$ for $j \in [1 \dots l]$, but not all inequalities are equalities. Similarly, the expression $n \prec \langle P_1, \dots, P_l \rangle$ returns true when $P_j = n.\text{data}().A_j$ for $j \in [1 \dots l]$.

The code first checks if the element that we are searching for is strictly in the left subtree (Lines 2-4) or in the right subtree (Lines 5-7). Of course, the subtrees are only considered if the appropriate marker bit is set. If both if statements fail (on Lines 2 and 5), then the current node has values P_1, \dots, P_l for the attributes A_1, \dots, A_l , respectively. If there is a node in the left subtrees with these values, then we recursively call the method on the left subtree. Otherwise, we simply return the current root node n . The algorithm finds the first node with the desired property and therefore is correct. At each step, the algorithm considers only the left or the right subtree and therefore it runs in logarithmic time relative to the size of the tree.

Algorithm 12 search1($n, A_1, P_1, \dots, A_l, P_l,$
 dir, l, i)

```

1: if  $\text{dir} = \text{asc}$  then
2:   if  $n.\text{left}() \neq \text{null}$  and  $n.\text{left}().m[i] = 1$  and
      $\langle P_1, \dots, P_l \rangle \prec n$  then
3:     return search1( $n.\text{left}(), A_1, P_1, \dots,$ 
       $A_l, P_l, \text{dir}, l, i$ )
4:   end if
5:   if  $n.\text{right}() \neq \text{null}$  and  $n.\text{right}().m[i] = 1$  and
      $n \prec \langle P_1, \dots, P_l \rangle$  then
6:     return search1( $n.\text{right}(), A_1, P_1, \dots,$ 
       $A_l, P_l, \text{dir}, l, i$ )
7:   end if
8:   if  $n \neq \langle P_1, \dots, P_l \rangle$  then
9:     return  $\text{null}$ 
10:  end if
11:  if search1( $n.\text{left}(), A_1, P_1, \dots, A_l, P_l,$ 
     $\text{dir}, l, i) = \text{null}$  then
12:    return  $n$ 
13:  end if
14:  return search1( $n.\text{left}(), A_1, P_1, \dots,$ 
     $A_l, P_l, \text{dir}, l, i$ )
15: end if
16: if  $n.\text{right}() \neq \text{null}$  and  $n.\text{right}().m[i] = 1$  and
    $n \prec \langle P_1, \dots, P_l \rangle$  then
17:   return search1( $n.\text{right}(), A_1, P_1, \dots, A_l,$ 
      $P_l, \text{dir}, l, i$ )
18: end if
19: if  $n.\text{left}() \neq \text{null}$  and  $n.\text{left}().m[i] = 1$  and
    $\langle P_1, \dots, P_l \rangle \prec n$  then
20:   return search1( $n.\text{left}(), A_1, P_1, \dots, A_l,$ 
      $P_l, \text{dir}, l, i$ )
21: end if
22: if  $n \neq \langle P_1, \dots, P_l \rangle$  then
23:   return  $\text{null}$ 
24: end if
25: if search1( $n.\text{right}(), A_1, P_1, \dots, A_l, P_l,$ 
     $\text{dir}, l, i) = \text{null}$  then
26:   return  $n$ 
27: end if
28: return search1( $n.\text{right}(), A_1, P_1, \dots,$ 
     $A_l, P_l, \text{dir}, l, i$ )

```

We will next consider the search method when $r = l - 1$. We will only show the code for when $\text{dir}_l = \text{asc}$, where the other case is symmetric. The pseudo-code is presented in Algorithm 13. Again n is initially the root node of the tree. The expression $\langle P_1, \dots, P_l \rangle \preceq n$ returns true when $P_j \leq n.\text{data}().A_j$ for $j \in [1 \dots l]$. The algorithm first checks to see if the resulting node is the right subtree (Lines 2-4). If not, then it checks the left subtree (Lines 5-7). If both options fail, then the algorithm checks if the current root node passes the condition. If it does, then it returns it (Line 11). If it does not, then it returns null (Line 9). Therefore, the algorithm

finds the correct node. The algorithm runs in logarithmic time on a balanced tree because it calls itself recursively on either the left or right subtree, but not both.

Algorithm 13 `search2(n, A1, P1, ..., Al, Pl, dir, r, i)`

```

1: if dir = asc then
2:   if n.right() ≠ null and n.right().m[i] = 1 and
    n ≤ ⟨P1, ..., Pl⟩ then
3:     return search2(n.right(), A1, P1, ..., Al, Pl, dir, r, i)
4:   end if
5:   if n.left() ≠ null and n.left().m[i] = 1 and
    search2(n.left(), A1, P1, ..., Al, Pl, dir, r, i) ≠
    null then
6:     return search2(n.left(), A1, P1, ..., Al, Pl, dir, r, i)
7:   end if
8:   if n ≠ ⟨P1, ..., Pl-1⟩ or n.data().Al ≥ Pl then
9:     return null
10:  end if
11:  return n
12: end if
13: ...

```

We will next show how the elements of the search tree can be efficiently retrieved in the order ⟨A₁ *dir*[1], ..., A_a *dir*[a]⟩ where *dir*[*j*] ∈ {asc, desc} for *j* ∈ [1 ... a]. The algorithm is presented in Algorithm 14, where the method will return only elements that belong to the set *S_i*. Note that the variables {*dir*[*i*]}_{*i*=1}^a are parameters to the method. However, they have been omitted from the algorithm in order to keep the code simpler and more understandable.

Algorithm 14 `ordered_next(n, i)`

```

1: if dir[a]=asc then
2:   n' ← next(n, i, false)
3: end if
4: if dir[a]=desc then
5:   n' ← previous(n, i, false)
6: end if
7: if n' ≠ null and  $\bigwedge_{i=1}^{a-1} (n.A_i = n'.A_i)$  then
8:   return n'
9: end if
10: if a = 1 then
11:   return null
12: end if
13: n' ← search(A1, n.A1, ..., Aa-1, n.Aa-1,
    dir[a-1], a-2, i)
14: return next_up(n, n', i, a-1)

```

Note that we have used *n.A_j* to denote the value of the *j*th attribute of the element that is stored in node *n*. The method calls the *next_up* method, which in turn can

call the *next_down* method. The methods are presented in Algorithms 15 and 16, respectively.

Algorithm 15 `next_up(n, n', i, l)`

```

1: if n' = null then
2:   if l = 1 then
3:     return null
4:   end if
5:   n' ← search(A1, n.A1, ..., Al-1, n.Al-1,
    dir[l-1], l-2, i)
6:   return next_up(n, n', i, l-1)
7: end if
8: return next_down(n', i, l)

```

Algorithm 16 `next_down(n, i, l)`

```

1: if n = null then
2:   return null
3: end if
4: if l = a then
5:   return n
6: end if
7: n' ← search(A1, n.A1, ..., Al, n.Al, dir[l+1], l, i)
8: return next_down(n', i, l+1)

```

Consider first Lines 1-6 of the method *ordered_next*. The code first checks whether the next node *n'* relative to the order ⟨A₁ *dir*[1], ..., A_{a-1} *dir*[a-1]⟩ has the same values for the attributed A₁, ..., A_{a-1} as *n*. If this is the case, then only this node needs to be returned. If this is not the case and *a* = 1 (Lines 10-12), then a “next” node does not exist and the method returns null. If this is not the case and *a* > 1, then Line 13 of the code looks for the first node that has the same value as *n* for the attributes {A_i}_{*i*=1}^{a-2} and a value for the attribute A_{a-1} that is right after the value of the attribute A_{a-1} for *n*. In Line 14 of the code the *next_up* method is called with the element that is found in the previous line, where the value for *n'* is null when such an element does not exist.

The method *next_up* looks for a node *n'* that has the property that $\bigwedge_{i=1}^{l-1} (n.A_i = n'.A_i)$ and the value for A_{*l*} of *n'* is right after the value for A_{*l*} of *n*. When this is the case, then the *next_down* method is executed. It finds the first element in the search tree for which $\bigwedge_{i=1}^{l-1} (n.A_i = n'.A_i)$ holds. This is indeed the node that needs to be returned. When *n'* does not have the desired property, *l* is decremented by 1 and *next_up* is called recursively. If *l* becomes equal to 0, then a “next” tuple does not exist and the method *next_up* returns null. Since the *next* method goes up and down a path in the three, its running time is logarithmic.

IV. CONCLUSION AND FUTURE RESEARCH

We introduced MB-trees and showed how they are beneficial for accessing predefined subsets of the tree elements. MB-trees

use marker bits, which add only light overhead to the different operations and do not change the asymptotic complexity of the operations. An obvious application of MB-trees is merging search trees by removing redundant data, which can result in faster updates because fewer copies of the redundant data need to be updated. In addition, we showed how elements in different orders can be retrieved from a search tree. Again, the application is index merging because fewer indices can efficiently answer the same set of queries.

One obvious area for future research is showing that the algorithm for retrieving all the elements of a search tree that belong to a set S_i in a non-trivial order takes time that is proportional to the size of the tree. Another contribution would be to present algorithms that extend our algorithms to secondary storage structures, such as B and $B+$ trees.

REFERENCES

- [1] G. M. Adelson-Velskii and E. M. Landis, "An Algorithm for the Organization of Information," *Soviet Math. Doklady*, vol. 3, pp. 1259–1263, 1962.
- [2] A. Andersson, "Balanced search trees made simple," *Workshop on Algorithms and Data Structures*, pp. 60–71, 1993.
- [3] R. Bayer and E. McCreight, "Organization and Maintenance of Large Ordered Indexes," *Acta Informatica*, vol. 1, no. 3, 1972.
- [4] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms*. McGraw Hill, 2002.
- [5] L. Stanchev, "Efficient Access to Non-Sequential Elements of a Search Tree," *The Third International Conference on Advances in Databases, Knowledge, and Data Applications, DBKDA 2011*, pp. 181–185, January 2011.
- [6] L. Stanchev and G. Weddell, "Saving Space and Time Using Index Merging," *Elsevier Data & Knowledge Engineering*, vol. 69, no. 10, pp. 1062–1080, 2010.

Transactional Composition and Concurrency Control in Disconnected Computing

Tim Lessner*, Fritz Laux*, Thomas Connolly†, Malcolm Crowe†

*Fakultät Informatik, Reutlingen University, Germany, name.surname@reutlingen-university.de

†School of Computing, University of the West of Scotland, name.surname@uws.ac.uk

Abstract—Composition of software components via Web technologies, scalability demands, and Mobile Computing has led to a questioning of the classical transaction concept. Some researchers have moved away from a synchronous model with strict atomicity, consistency, isolation and durability (ACID) to an asynchronous, disconnected one with possibly weaker ACID properties. Ensuring consistency in disconnected environments requires dedicated transaction support in order to control transactional dependencies between software components and provide a scalable concurrency control mechanism. This paper contributes a simple expression language using Boolean operators to define transactional dependencies and further provides rules to derive an execution semantics that could be exploited by a transaction manager to control the interaction. This work also discusses the use of data classes that demarcate data based on concurrency control related aspects and apply a certain concurrency control mechanism to each class. Such a classification allows better trade-off between consistency needs and the overhead caused by the concurrency control mechanism.

Index Terms—Transaction Management; Disconnected Transaction Management; Advanced Transaction Models; Concurrency Control; Optimistic Concurrency Control; Semantic Concurrency Control

I. INTRODUCTION

The last few years have shown a need for mechanisms and technologies to easily compose scalable and everywhere available applications. Service Oriented Computing (SOC), specifically the composition of services as well as the automated execution of business processes, Cloud Computing, using infrastructure services via the Web in a pay as you need manner, and the growth in Mobile Computing solutions, available everywhere, all represent aspects of this development. In [1] we have presented our first idea of an optimistic and disconnected transaction model that supports local autonomy of software components – a key characteristic of SOC and Mobile Computing.

The challenge for transaction management is to provide scalable mechanisms that ensure that data stays consistent across local boundaries between departments or even companies while the boundaries of transactions grow with the integration of new services –and their software components– to build new composite applications. In its widest form, data must be maintained consistently across several world wide distributed physical nodes due to availability demands and thus scalability is a key issue. Data often needs to be modified even if the connection is temporarily lost.

To facilitate loose coupling and increase autonomy, data should be modified in a disconnected and not in an online manner. This means that the set of proposed modifications to data is prepared offline and written back using a different set of transactions and not the same transactions that have been executed to read the data. This copes also well with the asynchronous message exchange that takes place in such an architecture. Data access is asynchronous too. In such an architecture, the traditional mechanism to keep locks on a database until the transaction has finished is no longer practical for the entire interaction. A locking isolation protocol, for instance, where other concurrent transactions read only committed results is not reasonable as it leads to long blocking time caused by the governing application's duration and the asynchronous message exchange.

Mobile Computing requires solutions for offline data processing despite the fast distribution and coverage of the mobile Web. Disconnected situations are frequent and users should be able to keep their data locally and synchronise the modifications back afterwards. Essentially, the circumstances in Mobile Computing are similar to that of SOC in that autonomy of software components is required including autonomy over the data they process.

A disconnected approach overcomes this challenge at the price of weakened isolation. The drawback of a weakened isolation is that other transactions can read pending results, which increases the danger for data to become inconsistent. To ensure consistency a validation must take place between (i) the phase a component (application) reads and modifies data locally and (ii) the phase the modifications are eventually written – merged with the database. Any transaction that is allowed to make an unverified change to data must specify a compensation transaction for restoring consistency if the process needs to be semantically undone later. For transactions that cannot specify a compensation it is therefore prohibited to make unverified changes.

Replication mechanisms that intend to increase the availability of data must scale and ensure that consistency of the different replicas is at least achieved eventually. An Eager replication [2] mechanism does not scale for highly replicated systems whereas lazy replication does, at the price that modifications of a transaction are not synchronised within the boundary of that transaction. Combining eager replication with a “master-slave” dissemination protocol scales and replicas can be made to converge within the transaction boundary [2].

But, in general, designing a highly replicated system requires a trade-off between the costs for consistency and scalability and thus availability [3], [4]. For some data, a mechanism for eventual convergence of replicas is sufficient, however, other data might require a much stronger consistency, possibly serializability. Some applications possibly require real-time behaviour, others may live with moderate availability. Recent research [3] shows that adaptive concurrency control based on a classification of data, leads to a better cost benefit ratio than one concurrency control to fit all needs. Thus, these considerations lead to a spectrum of different concurrency control protocols starting with no consistency at all and ending up with serializability.

The above review identifies the following characteristics of transactions that lead to complex transaction management:

- 1) Composition (dynamic): heterogeneous and autonomous software components represented as services are loosely coupled to create new composite applications. Due to the composition, transactional dependencies among the components arise.
- 2) Long-living nature: whereas the actual operations to read data and write modifications to the database are short lived¹, the overlying application (e.g., workflow) has a long-living nature. The result is a discrepancy between the time and the extent to which the physical operations (reads and writes) need ACID and the time the governing long-living application does. Isolation and consistency apply to both the read and the write phase of the application. However, compensation defeats durability.
- 3) Replication: nodes are physically distributed and replication must ensure that replicas converge depending on the data's semantics.
- 4) Disconnection: mobility requires disconnected transaction processing because of physical unavailability of the network connection. Also, to facilitate loose coupling, increased local autonomy is helpful to ease composition and due to an asynchronous message exchange, the set of proposed modifications to data is prepared offline for a later transacted sequence of operations on separate database connections. Notice, the long-living nature requires disconnected processing of data too, because keeping locks for the entire duration of the governing application would significantly reduce the concurrency.

In this paper, the focus is not on replication. We focus on composite, long-living, and disconnected transactions based on our first considerations published in [1]. We have removed much of the terminological overhead, and clearly demarcate concepts. One part specifically focuses on the transactional composition of transactions in a formalised manner using Boolean algebra (see Section III-B). We also added a section (see Section III-D) that deals with the classification of data based on the data's concurrency control (CC) properties and

¹Molina et al. [5] state that to precisely classify a transaction as either short or long living is complicated. They define a transaction as long living if to lock data for the transaction's duration leads to an undesired decreased concurrency or even thrashing.

a different CC mechanism is applied to each class. The classification has been inspired by [3], [6]. We present a simple reference architecture (Section II), where we also introduce our idea of a “Disconnected Component”. In Section IV, we present some existing transaction models and mechanisms as part of the related work, before Section V concludes this paper and outlines our future work.

II. ARCHITECTURE

We start by introducing a simple reference architecture (see Figure 1) that consists of three levels: database, middleware, and application level.

Database systems reside at the database level and they might be highly distributed as well as replicated. Also heterogeneous database federations, so called Multi-Database Systems (MDBS) [7], can exist. In an abstract view, the entire database level must be represented as one MDBS. Moreover, since mobile applications are also part of our architecture and mobile applications can use the mobile platform's database to increase their local autonomy to cope with frequent disconnections, the database can be logically also considered as a “Mobile MDBS” according to [8].

The middleware provides data access and owing to the assumed disconnected and asynchronous nature, data is read, copied, modified and synchronised back in a sequence of different independent transactions. A component (see Section II-A) starts a transaction (or a number of transactions) to read the data, disconnects, and locally modifies the data. After the modifications have been performed locally, the component sends just the changes back and based on these changes the middleware executes transactions to write the modifications. The middleware is allowed to use locks for reading and writing data from or to the database. Transactions in middleware and database layers are short-lived and locking is feasible, while retaining locks for the entire duration of the governing application is not practical.

The middleware plays a key role in this architecture. On the one hand it provides data access, on the other hand it has the role of a coordinator. Long running and hierarchically structured transactions involving many distributed, loosely coupled, heterogeneous, and autonomous systems require coordination. Also, interactions with external applications require transactional consistency. However, the middleware cannot enforce consistency of external systems. Often components are hosted by the middleware and composed together to build new applications as in SOC.

Application level refers to any component that implements concrete business logic. Components may also ship with their own, possibly replicated, database to increase their autonomy (see the “Composition Autonomy Pattern” in [9], for example). Mobile components are part of our reference architecture too. From a transactional point of view we do not believe that mobile components differ from stationary ones because both types of components have to cope with disconnection. For the remainder of this paper, we refer to a disconnected component

as a software component or just a component if the context is obvious.

The outer surrounding box in Figure 1 represents a transactional integration of components across the different levels.

A. Disconnected Component

An application consists of several software components that are either locally or remotely accessed. A component starts a number of transactions whereas the set of transactions to read the data is different from the set of transactions to eventually write the modifications. A component is defined to be either in its read, disconnected and working, or write phase, which is similar to Meyer's "Command Query Separation" pattern [10]. Figure 2 illustrates the idea.

To ensure consistency, the transition from disconnected and working to write requires validation. This validation is performed by a transaction manager but a component must ensure that the set of changes is passed to the transaction manager indicating at least the values or version read, and the new values of data. Technically, this means a component needs some book keeping functionality as defined in the Service Data Object (SDO) framework [11], for instance.

Components are allowed to call other component(s) to read data or to pass their modifications. Components used within a phase are said to be within this phase. Inside a component, the execution of flat transactions or calls to other components is not arbitrary and the implementation reflects an order in which execution takes place. We require a component to define this order if not defined elsewhere, e.g., by a workflow. Using another component is also represented as a transaction because these calls are transactional too. Their state is made persistent by writing the messages a component sends and receives to non-volatile memory. This concept is also known as persistent queuing. If each component specifies an order of execution, a composition order is the union of all the components' orders. If the read and write phases are separate, each component has to define two separate orders, one for the read and one for the write phase.

The Sagas [12] model discusses the notion of compensation to semantically undo the effects of long running transactions. Compensation has been introduced to cope with the requirement for weak isolation that arises if several transactions form a long living process but each of the sub-transactions is allowed to commit. Under this circumstance other transactions may read pending results. In case the transaction aborts, already committed sub-transactions need to be undone, which is only possible by executing a compensation, e.g., to cancel a flight is the compensation of booking a flight. If a sub-transaction is not compensatable, it is not allowed to unilaterally commit. The sub-transaction needs to pre-commit (promise) and wait for the global commit. If the global outcome is abort, the sub-transaction needs to rollback (there is no compensation). Compensation is discussed in Section III and here it is sufficient to introduce a compensation handler that points to another component that can implement the compensation.

DEFINITION II.1: (Disconnected Component):

A disconnected component is defined as a quintuplet $dc := (T^r, T^w, O^r, O^w, dc^{-1})$ with

- 1) a set of transactions T^r to read data,
- 2) a set of transactions T^w to write modifications,
- 3) a partial order for reading: $O^r = (V^r, E^r)$ with $V^r \subseteq T^r$ and the set of edges E^r is defined as $\forall t_n, t_o \in V^r : t_n \rightarrow t_o \Leftrightarrow e \in E^r$ with $e = (t_n, t_o)$. Transactions that do not belong to the subset V^r are said to be free transactions and hence can be executed in any order.
- 4) another partial order for writing: O^w be another partial order $O^w = (V^w, E^w)$ with $V^w \equiv T^w$ and the set of edges E^w is defined as $\forall t_n, t_o \in V^w : t_n \rightarrow t_o \Leftrightarrow e \in E^w$ with $e = (t_n, t_o)$. Transactions that do not belong to the subset V^w are said to be free transactions and hence can be executed in any order.
- 5) a compensation handler of dc .

If dc executes transactions T^r , it is in its read phase and if it executes transactions T^w it is in its write phase. Between these phases dc is in its disconnected and working phase. The write phase is not required for components that only read data.

The next section introduces the disconnected transaction model and provides a detailed definition for a transaction. A recursive model for transactional composition is the subject of this section too. The composition of disconnected components is eventually a composition of transactions. The resulting transactional dependency between two components is important and we provide a general applicable notion for them (see Section III-B).

III. TRANSACTION MODEL

The transaction model presented in this section is structured as follows: first, a general definition for a disconnected flat transaction is provided. The next part focuses on the composition of flat transactions and how to formalise the resulting transactional dependencies based on a Boolean expression. Based on such an expression, the third part discusses how to derive the execution structure of a composite transaction. The fourth part discusses different concurrency control protocols with a focus on optimistic and semantics based concurrency control (CC) mechanisms and defines different data classes according to the discussed CC mechanisms. This is the "Data View" of this model and its purpose is to demarcate data based on CC properties.

A. Disconnected, Flat Transaction

Our transaction model starts with the smallest unit: a flat transaction, the key building block. The following definition is based on the definition by Weikum and Vossen ([4], p.46) for a flat transaction.

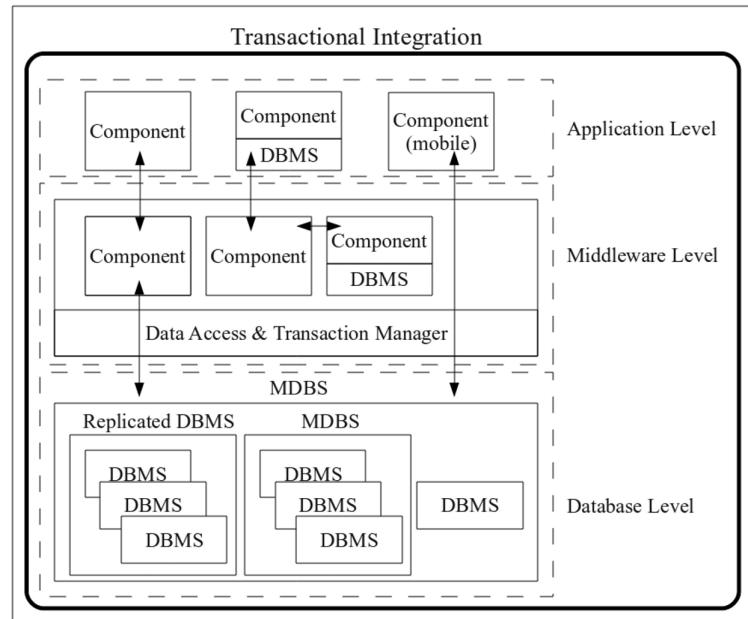


Fig. 1: Architecture

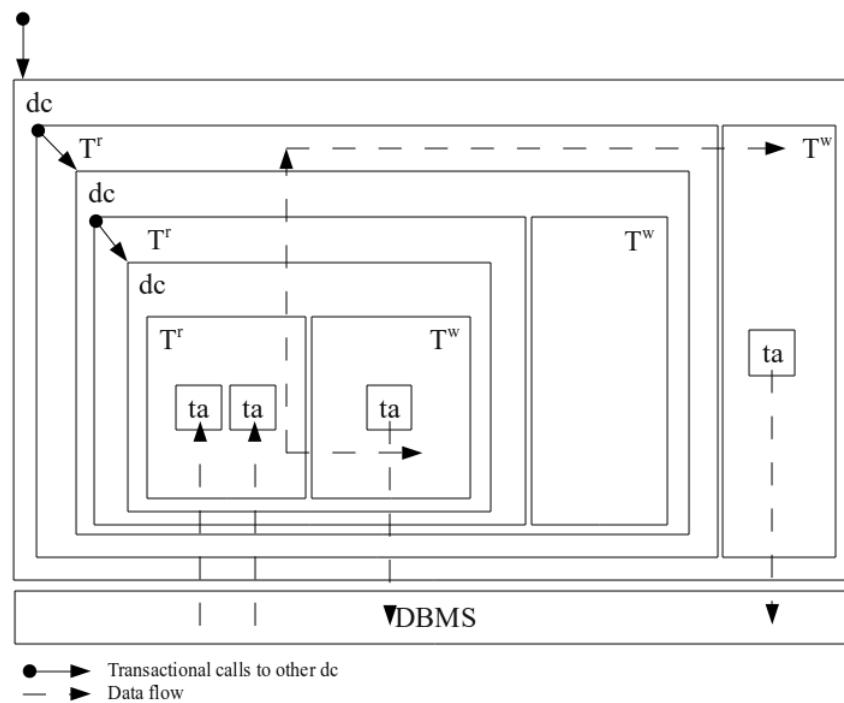


Fig. 2: Structure and composition of a *dc*. Dashed arrows show the data flow between the phases (shown by their related sets of transactions T^r , T^w) and upwards in the composition.

DEFINITION III.1: (Disconnected Flat Transaction I):

- 1) Let t be a flat transaction that is defined as a pair $t = (op, <)$ where op is a finite set of steps of the form $r(x)$ or $w(x)$ and $< \subseteq op \times op$ is a partial order.
- 2) A transaction is either in its reading (p1), disconnected and working (p2), validating (p3), or writing (p4) phase. The write phase is not required for read only transactions.

Section “Architecture” states that a software component uses different transactions to read and write data. This, however, requires a validation to ensure consistency. These phases have been introduced to avoid locking and support disconnection. They are similar to what is known as Optimistic Concurrency Control (OCC) [13], [14]. The only formally motivated difference is the explicit disconnected and working phase (p2). Usually, in the original OCC model, the actual work is done within the read phase. *“The body of the user-written transaction is in fact the read phase [...]”* [13]. Since we do not believe that Kung’s reduction represents the actual phases of a disconnected transaction, these phases are made more explicit in our model. Later (see Section III-D) we introduce data classes and apply a certain CC mechanism. In this section also the phases, especially validation, are thoroughly discussed.

The next step is to provide a general notion for the composition of flat transactions. The idea is to consider a set of flat transactions and define their composition by a Boolean expression and based on this expression to derive an execution structure. The last step is to transform the model into a recursive model. With this recursive model in hand we have a general notion for the transactional composition or integration of software components too.

B. Composition of Transactions and Transactional Dependencies

In complex transaction processing scenarios, such as distributed or workflow transactions, the “Degree of Transaction Autonomy” [15] can be expressed by transactional dependencies.

One well known example is a distributed commit where, for example, two transactions have to either bilaterally commit or abort. This creates a transactional dependency between the transactions so that their autonomy is weak (becomes part of interpretation). In another situation, however, it might be possible that a transaction is allowed to commit even if other dependent transactions abort and the autonomy of this first transaction is high because it is independent of the others’ outcome.

Consider, for example, a transactional workflow like the booking of a journey with several acceptable outcomes. The workflow consists of the booking of a hotel, a flight, and the booking of either a train or a car for a trip at the destination. So the satisfaction is that the booking of the hotel and the flight must succeed, whereas for the booking of the train or

the car the allowed outcome is either the first or the second. Thus, it is required that only the first two transactions commit, whereas for the later ones only one is allowed to commit. In an auction, for example, where a user wants to purchase three items, it could be acceptable to buy just one, two, or all of them. Both examples show that applications can have different acceptable outcomes, a so called satisfaction.

Transactional dependencies have been investigated in the domain of nested and advanced transaction processing [7], [16], [17] where a parent, for example, depends on the commit or abort of its children – a property known as “vitality of a child”. The opposite is known as “dependency of a child”; that is, a child depends on the commit of its parent. Notice, vitality and dependency affect the A of ACID.

The next section analyses transactional dependencies defined by a Boolean expression. We believe this is a useful reduction that makes transactional dependencies computable, and offers an execution structure.

1) *Satisfaction of a Transaction:* To represent transactional dependencies it is sufficient to define a satisfactory (acceptable) outcome for a set of transactions. For example, one satisfaction sf for $T = \{t_1, t_2, t_3\}$ could be $sf_1 = (c(t_1), c(t_2), c(t_3))$ another $sf_2(c(t_1), a(t_2), a(t_3))$ with $c:=$ commit and $a:=$ abort. Now, if after an execution of T (assumed, for example, a parallel one) one of the possible outcomes matches with the pre-defined outcomes sf_1 or sf_2 , T can be committed. If not, T needs to be aborted and all (committed) $t \in T$ must be rolled back or compensated if committed already.

Another way of representing a satisfactory outcome uses Boolean expressions and interpret true as commit and false as abort. For example, the satisfactory outcome sf for $T = \{t_1, t_2, t_3\}$ could be $sf = (t_1 \wedge t_2 \vee t_3)$. For $T = \{t_4, t_5, t_6, t_7\}$ the satisfactory outcome could be $sf = (t_1 \vee (t_2 \wedge (t_3 \vee t_4)))$. Boolean expressions can express a nested behaviour, which is a key requirement to model transactional dependencies. Another benefit of Boolean expressions is that they can be verified. Boolean expressions would at least allow to compute the “Satisfiability” (SAT) or “Validity” of the expression itself. This information makes it easier to reason about the correctness of transactional dependencies.

DEFINITION III.2 (Satisfaction of transactions):

- 1) Let $T_k = \{t_1, \dots, t_n\}$ be a finite set of t where $T_k \subseteq T$ is a subset of the superset T of all transactions.
- 2) The set of satisfactory outcomes defining all acceptable outcomes for T is defined as: $SF(T) = \{sf_1(T_1), \dots, sf_j(T_k)\}$ with $sf(T_k) = expr$.
- 3) Let
 - $expr := (expr) op (expr)$
 - $expr := c(t) \mid a(t)$
 - $op := \wedge \mid \vee \mid \oplus \mid p_l \mid p_r$
- 4) $v : TRUE \leftrightarrow c(t)$ and $v : FALSE \leftrightarrow a(t)$ with $c(t)$ being the commit and $a(t)$ the abort of a transaction.
- 5) Let $OUT(T) = \{out(t_1), \dots, out(t_i)\}$ be the set of the

- atomic transactions' outcomes after transaction processing with $out(t) \in \{c(t), a(t)\}$.
- 6) Let $\sigma : (OUT(T), sf(T)) \mapsto s(T)$, with state $s(T) = c(T)$ or $a(T)$, commit or abort of a set of transactions T . Each $out(t_i)$ is mapped to the according occurrence in each $expr$. Validation function σ validates each $expr$.

Concerning the number of allowed operators op compared to the number of Boolean operators, the number of allowed operators is limited to “AND” \wedge , “OR” \vee , “XOR” \oplus , and the two projections p_l, p_r . See Table I for a complete overview.

- 1) Operator \wedge : The logical “AND” is a common case and only if both transactions commit, the result is committed.
- 2) Operator \vee : The logical “OR” represents the situation where it is sufficient if at least one of the transactions commits.
- 3) Operator \oplus : The logical XOR where exactly one transaction is allowed to commit.
- 4) Operators p_l, p_r are projections and discussed as well as defined (see Definition III.3) below.

Other Boolean operators are less reasonable with respect to transactional dependencies. Implication, for example, would mean that the abort of two transactions, i.e., $f \rightarrow f = t$ validates to true even if the transactional system's state has not changed. Generally, operators validating abort and abort (false and false) to true are less reasonable. Except the “XOR”, operators that validate commit and commit to false are less reasonable for the same reason. Generally, the semantics of the logic is that the abort of a transaction is not a correct result, even though it is consistent. Although the “XOR” operator is an exception, it is required because in some scenarios only one of two transactions is allowed to commit (book either the train or rent a car; buy either this item or the other).

The “NOT” operator is not listed in Table I. To define the satisfaction of a transaction as not commit (=abort) means the processing of the transactions is not intended at all. For example, the expression $expr = t_1 \wedge \neg t_2$ states that a commit of t_1 and an abort of t_2 is satisfying. This expression is equivalent to $expr = t_1$ because to start the execution of t_2 with the goal to let t_2 abort is not correct. To model a transaction with the intention to let the transaction abort is not reasonable. Even in a situation, for example, to test a system with the intention to throw an exception, the general semantics of a transaction requires the transaction to commit to throw the exception. The ambiguity with this operator is that for a single transaction t , the possible outcome is indeed $expr = t \oplus \neg t$.

To resolve this ambiguity and to comply with the correctness semantics of a transaction, the “NOT” operator is not allowed in expressions, but in case an expression needs to be optimised for validation and therefore transformed into a conjunctive or disjunctive normal form the “NOT” operator might be required. For instance, expression $a \oplus b$ can be transformed into the disjunctive normal form $a \oplus b = (a \wedge \neg b) \vee (\neg a \wedge b)$.

The function v maps true (T) to the commit of a transaction and false (F) to abort.

TABLE I: Operators for transactional dependencies.

#	t_i	t_j	\wedge	\vee	\oplus	p_l	p_r
1	c	c	c	c	a	c	c
2	c	a	a	c	c	c	a
3	a	c	a	c	c	a	c
4	a	a	a	a	a	a	a

To perform a validation $OUT(T_k)$ represents the set of outcomes of atomic transactions T_k . Based on these outcomes, the outcomes of composed transactions T_k are computable. Regarding σ it is important that the state after validation is actually pending since the final outcome might not be determinable yet as it possibly depends on the validation of other dependent T_l .

The projections p_l and p_r represent a transactional dependency where the outcome (result) of one transaction (operand) supersedes the other. For p_l the left argument supersedes the right, for p_r the right supersedes the left (see Table I).

Special cases of transactional dependencies have been investigated already and are known as “vitality” or “dependency” of a transaction. A transaction is said to be vital if its abort leads the parent transaction to abort too. Non-vital if the child's abort does not affect the parent. A transaction is said to depend on the parent, if the parent's abort leads the child to abort too. If not, the transaction is said to be independent.

DEFINITION III.3 (Projection operators): Let the left argument of a projection be the parent and the right argument be the child. Then, according to the notions of vitality and dependency let projection p_l be a combination of non-vital and dependent and let p_r be a combination of vital and dependent.

Projection p_l is non-vital because even if the child (right operand) aborts the outcome is still commit. It is dependent because if the parent (left operand) aborts the global outcome is abort and the child must be rolled back. The interpretation of p_r is accordingly.

It is also possible to define an interpretation for the other operators according to the notions of vitality and dependency. Table II shows the possible combinations of vitality and dependency and how they map to the operators. Notice that only mixed outcomes are shown in the table as vitality and dependency relate to mixed outcomes only. The only exception –again– is the “XOR” operator because the behaviour is different in case both transactions commit, as discussed above, and vitality as well as dependency are not directly applicable to the “XOR” operator. Notice, our model subsumes the notions of vitality and dependency.

Strictness (see Table II), which is given if one of the concepts is dependent or vital, describes the autonomy of a transaction and a strict operator represents a weak autonomy whereas a non-strict operator represents autonomy. Although vitality and dependency are not applicable to “XOR”, it is strict because the transactions are abort dependent. If both

can commit, both have to abort.

Concerning the validation of an expression, the projection operators have an interesting property. For example, consider the expression $expr = t_1 \text{ } p_l \text{ } (t_2 \vee (t_3 \wedge t_4))$. If $expr$ is reduced to $expr = expr_1 \text{ } p_l \text{ } expr_2$ where $expr_1 = t_1$ and $expr_2 = (t_2 \vee (t_3 \wedge t_4))$ it can be shown that in each possible distinct permutation of true and false, the outcome of $expr_1$ supersedes $expr_2$ (see Table I). Thus, there is a dominant part and projection operators (both operators show this behaviour) should be validated first and the remaining non-dominant part needs not to be considered for validation.

Based on the observations so far, the precedence, associativity, and commutativity can be defined as in Definition III.4.

DEFINITION III.4 (Precedence, Associativity, and Commutativity): The operator precedence is defined from high to low as follows: projections p_l and p_r , conjunction \wedge , disjunction \vee , and XOR \oplus . The associativity convention is that operators associate to the right.

LEMMA 1: The projections p_l and p_r are not commutative.

Proof: By contradiction. Suppose the projections p_l and p_r are commutative. Given an expression $expr_i \text{ } p_l \text{ } expr_j$ and let $expr_i$ validate to true and $expr_j$ to false, hence the global result is commit (true). Due to commutativity the global result is commit too, if $expr_j$ commits and $expr_i$ aborts. As shown in Table I this is not true and the result is abort in case $expr_i$ aborts and $expr_j$ commits. Since the proof for $expr_i \text{ } p_r \text{ } expr_j$ is equivalent, the projections are not commutative. ■

Regarding projections and validation, an expression can be reduced and nondominant parts can be skipped for validation. Figure 3 shows two example syntax trees. In the first example the expression $expr = (t_1 \vee t_2) \wedge (t_3 \oplus (t_4 \text{ } p_l \text{ } (t_5 \text{ } p_r \text{ } t_6)))$ is reduced to $expr' = (t_1 \vee t_2) \wedge (t_3 \oplus t_4)$. Part $(t_5 \text{ } p_r \text{ } t_6)$ is nondominant and does not affect the global outcome. Similarly, the expression $expr = (t_1 \vee t_2) \text{ } p_r \text{ } (t_3 \oplus (t_4 \wedge t_5 \wedge t_6))$ is reducible to $expr' = (t_3 \oplus (t_4 \wedge t_5 \wedge t_6))$.

DEFINITION III.5 (Reducibility of Projections): In a projection the argument that is not projected is called non-dominant. For $expr_i \text{ } p_l \text{ } expr_j$ the non-dominant part is $expr_i$, for $expr_i \text{ } p_r \text{ } expr_j$ it is $expr_j$.

LEMMA 2: For validation, an expression $expr$ can be reduced by all nondominant expressions

Proof: By contradiction. Given an expression $expr_i \text{ } p_l \text{ } expr_j \neq expr_i$. From Table I it follows directly that this is not possible. ■

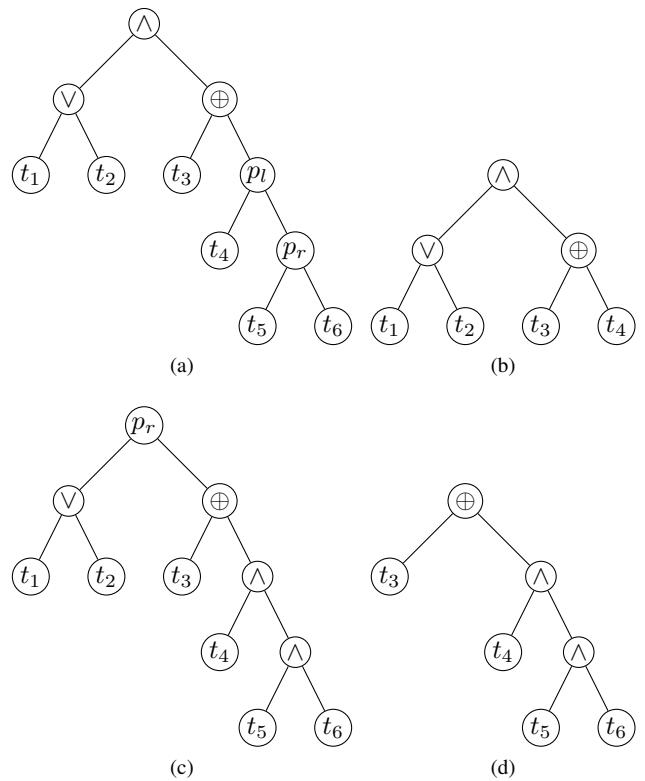


Fig. 3: (a) Complete and (b) reduced syntax tree of $expr = (t_1 \vee t_2) \wedge (t_3 \oplus (t_4 \text{ } p_l \text{ } (t_5 \text{ } p_r \text{ } t_6)))$. (c) Complete and (d) reduced syntax tree of $expr = (t_1 \vee t_2) \text{ } p_r \text{ } (t_3 \oplus (t_4 \wedge t_5 \wedge t_6))$

So far, a representation for transactional dependencies using a reduced Boolean algebra has been introduced as well as the required validation rules. The next step is to define the “Execution Structure” with respect to atomicity.

C. Execution Structure - Atomicity

Operators link transactions and define their transactional dependency, but to ensure atomicity as defined by a satisfaction expression requires some additional measures. For example, what is additionally required if two transactions state a strong dependency indicated by the \wedge operator? What is required if two transactions are linked by a projection operator? The idea here is to derive the “Execution Structure” from the sf function.

For example, given the expression $sf = t_1 \wedge t_2$ requires an atomic commitment, a 2PC for instance, because both have to commit or abort. Usually, either t_1 or t_2 plays the role of the 2PC coordinator or an additional instance fulfills this role. But, in either case this additional measure of a coordinator is neither represented by t_1 nor by t_2 . The operator, the \wedge in this example, indicates the measure that needs to be taken, namely, to ensure that both transactions belong to the same composite transaction, which only commits if both children do. Technically, a transaction manager needs to create a context (a composite or boundary) to coordinate the interaction among

TABLE II: Operators, Vitality, and Dependency.

<i>op</i>	<i>t_i</i>	<i>t_j</i>		vitality, dependency	strictness
\wedge	c	a	a	<i>t_j</i> is vital for <i>t_i</i>	strict
\wedge	a	c	a	<i>t_j</i> depends on <i>t_i</i>	
\vee	c	a	c	<i>t_j</i> is non-vital for <i>t_i</i>	non-strict
\vee	a	c	c	<i>t_j</i> is independent from <i>t_i</i>	
\oplus	c	c	a	-	strict
\oplus	c	a	c	-	
\oplus	a	c	c	-	
<i>p_l</i>	c	a	c	<i>t_j</i> is non-vital for <i>t_i</i>	strict
<i>p_l</i>	a	c	a	<i>t_j</i> is dependent from <i>t_i</i>	
<i>p_r</i>	c	a	a	<i>t_j</i> is vital for <i>t_i</i>	strict
<i>p_r</i>	a	c	c	<i>t_j</i> is independent from <i>t_i</i>	

transactions. Transactional context is a rather implementation related concept to ensure that each transaction is within the same boundary, i.e., has the same context, e.g., transaction id or possibly shared data. It is further possible to define an atomic completion (Atomic Sphere [18]) for a context. The boundary in turn demarcates a composite of transactions and may be a sub-transaction in a larger context. This should become clear if an application involving several transactions is modelled as a recursive tree.

So, deriving the composite structure means deriving the atomicity related execution semantics for the operators and the result is the actual execution structure itself. For example, $sf = t_1 \wedge t_2$ technically (which measures need to be taken to ensure atomicity) means $TA = (t_1, t_2)$ where TA is a composite requiring an atomic outcome.

Before providing a definition for the derivation, the idea for the definition is motivated first.

One motivation for a demarcation of sub-transactions is as follows: Let us assume that an application is not divided into sub-transactions and represented as one transaction instead. Then the entire transaction needs to be processed as one unit of work, which increases the blocking time (in case locking schema concurrency control protocols are used) of resources due to the fact that more things need to be processed within one atomic step. In case OCC is used resources are not blocked, however, the problem is still the aforementioned discrepancy between the time the transaction lasts and the probability of conflicts (the longer the transaction lasts, the higher the conflict probability). If sub-transactions could be demarcated though, these units could be processed individually and interleaved with others to increase the concurrency. The price in turn is that to individually process sub-transactions leads to a weakened atomicity because some sub-transactions possibly have committed before the entire global transaction has terminated. In case the global transaction aborts, already committed sub-transactions need to be compensated. This is only possible if compensations for the already committed sub-transactions exist². Compensation can be interpreted as a necessity to conform to reality like the cancellation of a

²Depending on the isolation (open or closed, see next section) a child's result might be visible to all transactions or just the parent. In the latter case the commit of the parent publishes its children's results.

booked flight. Here, we are only interested in motivating our idea, which is based on the notion of compensation, so just the basics are discussed. For further details on compensation, see Garcia Molina [12] and Leymann [19].

A sub-transaction (ST) is allowed to unilaterally commit if: (i) a compensation for ST exists. If the component is a composite, a compensation for each member must exist; (ii) in case some members have no compensation, these members must be free of effects (e.g., read-only transactions are free of effects). If neither (i) nor (ii) holds, then a unilateral commit is not possible as long as the global outcome has not been determined. The outcome is determined if the satisfaction has been validated, but as long as the outcome has not been determined, the ST is in state "pre-commit" – a promise to eventually commit. An abort is always final and in case the global outcome is abort (satisfaction is false), each committed or pre-committed ST must compensate or abort and release its state ("State Release"). In case the global outcome is commit, each ST can retain its state ("State Retention"), also the aborted ones.

Applying the idea of "State Retention" and "State Release" to the operators, it follows that only the non-strict operator (see Table II) retains its state independent of the global outcome. The reason is the "OR" explicitly represents a transactional dependency where one of the transactions is allowed to abort although the global outcome is commit. This in turn means that transactions linked with the non-strict operator do not require to belong to the same **directly** higher ordered transaction (composite). Notice that due to the recursive nature they might be part of some higher ordered transaction (composite). Regarding strict operators, a situation is given where one of the sub-transactions commits even though the outcome is abort. Therefore, the committed transaction has to release its state and strict operators always require the same directly higher ordered transaction (composite) for their linked transactions.

First we define a composite transaction as follows:

DEFINITION III.6: (Composite Transaction) Let $T_i \in \{ta, TA\}$ be either a flat transaction ta or a composite transaction TA where $TA := \bigcup_{i=1}^m T_i$.

Notice, this is a re-definition of t and T . From now on, ta represents a flat transaction as defined in Definition III.1. Since T might be either a flat transaction or a composite, the model is recursive.

Next, we define the derivation of a composite as follows:

DEFINITION III.7 (Derivation of the execution structure):

Given an expression $T_i \ op \ T_j$ with $op \in \{\wedge, \oplus, p_l, p_r\}$ it follows that T_i, T_j are part of a higher (parent) TA with $T_i, T_j \in TA$. TA is called a derived composite transaction.

EXAMPLE III.1: Given the following expression:

$$sf = ((ta_1 \wedge ta_2) \vee ta_3 \vee (ta_4 \wedge ta_5) \vee ta_6 \vee (ta_7 \wedge ta_8)) \oplus (ta_9 \wedge (ta_{10} \ pr_l \ ta_{11})).$$

Applying the rule from Definition III.7 the following structure is derived:

Due to the \oplus :

$$TA = \{TA_1, TA_2\} \quad (1)$$

Applying the rules on TA_1 :

$$\begin{aligned} TA_{1,1} &= \{ta_{11}, ta_{10}\}, TA_{1,2} = \{TA_1, ta_9\} \\ TA_1 &= TA_{1,2} \end{aligned} \quad (2)$$

Applying the rules on TA_2 :

$$\begin{aligned} TA_{2,1} &= \{ta_8, ta_7\}, TA_{2,2} = \{ta_6\} \\ TA_{2,3} &= \{ta_5, ta_4\}, TA_{2,4} = \{ta_3\} \\ TA_{2,5} &= \{ta_2, ta_1\} \\ TA_2 &= \{TA_{2,1}, ta_6, TA_{2,3}, ta_3, TA_{2,5}\} \end{aligned} \quad (3)$$

It is important to understand that none of the TA actually exists during the time the sf has been defined. A TA is a derived composite to ensure the atomic outcome of its components, its nature is purely technical, and it is created at the time the expression is validated.

Notice, in Example III.1 we can just write T instead of ta and leave the actual type (TA or ta) open because of the recursive nature.

The next step is to consider the situation in which a T_i exists more than once within the same sf . As said, the difference between a derived composite TA and a T is that TA is a transactional context, whereas a T exists with respect to a functional requirement. Assume, for example, given the following expression $sf = (T_1 \wedge T_2) \vee (T_1 \wedge T_3)$. After the derivation of the composite structure we have $TA_1 = \{T_1, T_2\}$ and $T_2 = \{T_1, T_3\}$. This means even if T_1 actually exists only once, the derived execution structure states it belongs to two different composites. This could be problematic, namely in case T_3 aborts T_1 has to abort too, and since T_1 belongs also to TA_1 , TA_2 has to abort too. Thus, T_2 is not independent of T_3 and both composites need to be aggregated.

Another reason is that a satisfaction sf may have the structure $sf = sf_1 \vee sf_2 \dots \vee sf_i$ to model different acceptable

outcomes – a key requirement especially when dealing with compensation. For example, $expr = (T_1 \wedge T_2) \vee (T_1 \wedge T_1^{-1} \wedge T_2 \wedge T_2^{-1})$ where T_i^{-1} represents a compensation of T_i . Due to the rule in Definition III.7 each \vee operator leads to the creation of its own boundary, which means there are eventually different representations for the same sf . There are different execution paths –this is why a satisfaction is required– but there is one execution structure only.

DEFINITION III.8 (Union of derived composites): Whenever two derived composites TA interleave, $TA_i \cap TA_j \neq \emptyset$, they have to be joined. Formally, the transitive closure is defined as $TA'^+ = \bigcup_{n>0}^o$ where $TA'_n = TA'_{n-1} \cup \{(T_i, T_k) \mid \exists T_j : (T_i, T_j) \in TA_{n-1} \wedge (T_j, T_k) \in TA_{n-1}\}, i \neq j \neq k$.

EXAMPLE III.2: Given the following expression:

$$sf = T_1 \wedge T_2 \vee (T_3 \vee T_4) \wedge (T_5 \vee T_6 \vee (T_7 \wedge T_8 \oplus T_9 \wedge T_{10} \ pr_l T_1)).$$

Applying the rule from Definition III.7 and III.8 the following structure is derived:

$$TA_1 = \{T_3, T_4, T_5, T_6, T_7, T_8, T_9, T_{10}, T_1\}, \quad (1)$$

$$TA_2 = \{T_1, T_2\}$$

$$TA_1 \cap TA_2 = T_1 \Rightarrow TA_x = \{T_1, \dots, T_{10}\} \quad (2)$$

As shown, the entire expression requires one derived composite TA_x to control the execution.

EXAMPLE III.3: Given the following expression:

$$sf = (T_1 \wedge T_2 \vee T_3) \vee (T_4 \wedge T_5 \vee T_6) \vee (T_1 \wedge T_7).$$

Applying the rule from Definition III.8 recursively the following structure is derived:

$$TA_1 = \{T_1, T_7\} \quad (1)$$

$$TA_2 = \{TA_{2,1}, TA_{2,2}\} \quad (2)$$

$$TA_{2,1} = \{T_6\}, TA_{2,2} = \{T_4, T_5\} \quad (3)$$

$$TA_3 = \{TA_{3,1}, TA_{3,2}\} \quad (3)$$

$$TA_{3,1} = \{T_3\}, TA_{3,2} = \{T_1, T_2\} \quad (4)$$

$$TA_1 \cap TA_3 = T_1 \Rightarrow TA_x = \{T_1, T_2, T_3, T_7\} \quad (4)$$

As shown, the entire expression requires two derived composites TA_1 and TA_x to control the execution.

Example III.3 defines the allowed outcomes and based on the sf , an execution structure is derived. The result are two derived composites TA_1 and TA_x , which means, that there are two independent composites. Assumed, the sf stands for a workflow, a representation of the workflow itself is missing. To represent this, a root composite TA^R needs to be created. So the actual execution structure of sf is: $TA^R := (TA_1, TA_x)$. Notice, a root composite is only required if there are more than

one derived composites after the last step of the recursion. We come back to the root composite briefly.

The next step is to provide a definition for an atomic completion protocol of a derived composite TA according to the “Open Nested Transaction Model” [20].

DEFINITION III.9 (Atomic completion): Each member T of a derived composite TA (except the root) has to retain its state in case the outcome of TA is commit. In case the outcome of TA is abort, T has to release its state by either rolling back if T is in state pre-commit or by compensation if T has locally committed already. T is only allowed to unilaterally and locally commit if it has a compensation T^{-1} . A transaction retains its state by a local commit, which is only required if no T^{-1} exists. TA' is aborted if all T have released its state in the opposite order.

The reason why the root has been excluded in Definition III.9 is that the root is only required if there are more than two derived TA left after the last step of recursively applying the rules. This can happen only if independent TA exist, which in turn means even in case one TA aborts the other may still commit and retain its state. TA^R serves a different need and technically it just reflects the results of all TA and hence all T .

An execution order of transactions has not been considered yet. The reason is that an execution order for transactions is usually given by the application’s implementation. Sometimes the order is applied externally to an application, for example, a workflow model could define the execution order. For our model it is just important that the order is accessible by the transaction manager to perform compensation if required. The order has been defined in Definition II.1.

Summarising this section, based on a sf for a set of transactions it is possible to derive an execution structure. The result are derived composite transactions that are technically required to ensure an atomic outcome (compensation is semantically equivalent to an atomic outcome) as defined by the sf .

The next section focuses on Data Classes and concurrency control related aspects. Whereas transactional dependencies affect the atomicity, consistency and isolation are subject of the next section.

D. Data Classes

As said in the Introduction already, disconnected data processing requires a non blocking mechanism that is able to ensure a strong consistency where the state of the data modified by a transaction has not changed during the execution – the so called isolation property –. Locking over the entire lifetime of the governing application is an inadequate solution. The problem with locking lies specifically in the long duration and the longer the transaction lasts, the longer the data is locked, and the less is the concurrency because other transactions cannot obtain any locks for this data.

Additionally, the longer the transaction lasts the higher is the probability that the transaction might fail. A proper time out setting must be considered as complicated because the database is left unaware about the intended duration of the application including the user interaction. Notice, even when using locking separately for the transaction(s) reading data and the one(s) writing modifications, inconsistencies might still arise because the transaction(s) to read and the transaction(s) to eventually write the modifications are logically independent. Another reason why locking is inadequate is for applications running on mobile devices. In mobile computing, network fragmentation and the resulting disconnection is considered as “being normal”.

OCC is a solution for this issue. However, the problem with OCC is the discrepancy between the time a transaction lasts and the probability of others wanting to modify the data. This is specifically crucial for update intensive entities –so called “Hot Spot Fields”.

To provide consistency for fields that are subjected to this asynchronicity, one solution is to exploit the semantics of fields where the conflict probability is high³. Hot spot fields are usually numeric and operations performed on these fields are thus arithmetic and usually commutative. Also, constraints on such fields are common too. For example, the current stock of an item or the number of available seats in a flight is limited. O’Neil [21] discusses the use of an “Escrow” data type for such fields. For “Escrow” fields, transactions can request a kind of a guarantee at their start time to successfully perform their modifications at the end. The guarantee can even depend on a constraint. If a guarantee has been granted, the transaction can continue processing in a disconnected mode and with the escrowed guarantee in hand, the transaction can commit successfully, assuming there are no other conflicts with non-Escrow fields. Concurrency on these fields is increased with such an approach because locks are required only for the time the guarantee request is processed (a consistent view on the “Escrow” field is indispensable during this time). Hence, even if O’Neil’s concept is pessimistic because actions are taken at the beginning of a transaction it fits well into a disconnected architecture.

Laux and Lessner [6] discuss a similar idea, however, instead of requiring guarantees at the beginning of the transaction their approach is optimistic (i.e., no measures at the beginning of the transaction) and performs a validation before writing. If the validation fails for a field, “Reconciliation” is possible if a “Dependency Function” for the conflicting entity is known and if the transaction wants reconciliation for the conflicting operations. “Reconciliation” describes the process of replaying an entire transaction or just the conflicting operation with the actual state of the database. To replay is only possible if the transaction or the conflicting operation is independent of further user input. This type of independence

³To precisely define such a probability requires statistical measures and usually a Poisson Distribution for conflicting transactions is postulated. See Kraska et al. [3] for a model for probability based consistency in replicated database systems.

is similar to the notion of “Logical Access Invariance” [22], [23].

A “Dependency Function” has the new value, the value read, and the current value of a field as possible input parameters and calculates a semantically correct state despite a conflict. For numeric fields, where operations are additions, subtractions, and multiplications, this function is a “Linear Dependency Function” (see [6] for further details).

The drawback of [6] compared to [21] is that in case of a constraint violation, the transaction has to abort possibly after a lot of work has been done. This is especially disadvantageous if the conflict rate is high on that field. Indeed, this reflects the optimistic behaviour and is therefore intended, but guarantees like O’Neil’s mechanism would be preferable despite their pessimistic nature especially if the conflict rate on this field is high.

Another difference is that Laux and Lessner’s approach has been designed for architectures where a “change-set” of proposed data is delivered by the client. The modifications are then passed to stored procedures or SQL transactions are generated and executed according to the changes. Change-sets, for example as used in Service Data Objects (see [11]), comply with the current nature of computing architectures where communication is asynchronous, message oriented, and disconnected. For example, also the work by Thomson and Abadi [24] is based on the observation that transaction processing has shifted away from a synchronous to an asynchronous mode. Asynchronous thereby means modifications are prepared offline and just the results of modifications or the new values are sent back to the database.

An additional difference between these concepts is that O’Neil foresees a classification based on a data type, but Laux and Lessner’s approach is on a transaction or even an operation base.

Both [6], [21] have in common that their mechanism applies to a certain kind of data only, usually numeric data. Even if the concept of a dependency function is a formal improvement compared to O’Neil, because it defines a precise property for “Escrow” fields (linear dependent in case of numeric types), it seems rather questionable to find such functions for non-numeric types. So, a non-blocking solution for all “Non-Escrow” fields is needed, which ensures strong consistency (e.g., serializability) already at the beginning of the transaction’s read phase, especially if the conflict probability on those fields is high. One option is to consider once again the semantics of those fields as well as the way the data is accessed. We believe that in many situations semantics are given so that the conflict rate is very low. For example, data belongs to a certain instance or node, or data is modified using insert semantics without even requiring an isolation level of “Repeatable-Read”.

But after all, some data might be just subject to conflicts. In a very limited way we believe it is possible to use locking, namely, only if to lock is “logically motivated”. For example, a salesman with his associated customers could have an owner role for these customer data and due to the ownership, locking

is justified. Locking in this context should not be considered as a mechanism to control concurrency (transparent for developers), it is rather a mechanism to enforce an ownership of data due to application specific issues (not transparent for developers).

In case a lot of data requires a strict locking, our model is not adequate. For example, frequent calculations over a set of data with an isolation level of repeatable read.

According to these previous considerations and according to [6], [21] the following properties are distinguishable (we will use the following abbreviations in Table III):

- properties of the data: does a constraint (*cons*) exist, is the type numeric (*num*)?
- the operations’ semantics: are operations on this field commutative (*com*), is a dependency function known (*dep*)?
- is user input independence (*in*) given?

Notice, a transaction is independent from the user input if a replay does not require any further user input. That is, the user’s intention is to execute the transaction despite an existing conflict with the same input. In case a complete transaction is about to be replayed, this property must hold for each of the transaction’s operations. In case a single operation is replayed only, this property must hold for the specific operation only.

Based on these properties five classes are defined (see Table III for an overview) where each class has a certain CC mechanism.

The second class “Reconcilable with constraint *RC*” introduces a conflict probability $P(X)$ and a threshold *th*. This is motivated by the consideration that to request a guarantee for an “Escrow” field leads to some overhead, which is only required if $P(X)$ is too high for validation to succeed. In case $P(X)$ is low, reconciliation should be used to reduce this overhead and to better comply with an optimistic nature. A definition for $P(X)$ is part of future work.

Eventually, Definition III.10 re-defines Definition III.1 and considers the different data classes.

DEFINITION III.10: (Disconnected, Flat Transaction II)

- 1) Let R, RC, NRE, NRO, NRL be data classes as defined in Table III
- 2) Let ta be a flat transaction that is defined as a triplet $ta = (op, \leq, u)$ where op is a finite set of steps of the form $r(x)$ or $w(x)$, $x \in \{R, RC, NRE, NRO, NRL\}$. And $\leq \subseteq op \times op$ is a partial order, and u denotes the user input.
- 3) A transaction is either in its reading (p1), disconnected and working (p2), validating (p3), or writing (p4) phase.

Definition III.10 makes the semantics of a write operation not explicit. The reason is that we aim for a classification based on data.

The issue with the last definition is that ta now technically becomes a composite transaction of the form TA with children

$ta_R, ta_{RC}, ta_{NRE}, ta_{NRO}, ta_{NRL}$ since to use a different CC mechanism means to divide a transaction into a maximum of five separate transactions (ta_R, \dots, ta_{NRL}). The operator of the satisfaction sf must be the \wedge operator. In case data of only one data class is modified the additional composite can be omitted.

For the remainder, we continue to refer to the composite with children ta_R, \dots, ta_{NRL} as ta and other composites as TA .

E. Consistency

To ensure serializability for lengthy or disconnected processing validation is required. In locking scheme concurrency control, Rigorous 2PL [4], [5], [25] ensures serializability but since locking is not an option for classes R, RC, NRE , and NRO an equivalent mechanism guaranteeing serializability is required. Therefore, for R, RC , and NRO an optimistic validation [14] needs to be performed to test whether a set of modifications (write-set) of a transaction intersects with a write-set or read-set of other concurrently executed transactions. In case the write-set of a validating transaction intersects with the read- or write-set of a concurrent transaction, one of the pairwise conflicting transactions has to be aborted. Such a validation ensures conflict serializability (CSR) [4]⁴. Reformulated the aforementioned means: if the write-set $WSet_i$ (data modified by a transaction i) intersects with the write-set WS_j or read-set RS_j (data read by a transaction) of another transaction, a conflict is present and one of the pairwise conflicting transactions should be backed out. Hence, to avoid intersections between two transactions ensures CSR. Following this observation a validation must ensure: $WS_i \cap RS_j = \emptyset \wedge WS_i \cap WS_j = \emptyset$.

Classes R and RC , however, have an interesting property and in case validation fails, reconciliation resolves the conflict and ensures semantic correctness. An algorithm for reconciliation is described in [6].

Transactions modifying escrow data NRE have to request guarantees during their read phase. Validation as discussed above is not required because the guarantee ensures consistency already. So, during the validation phase it just has to be ensured that a guarantee has been granted during the read phase. For further details it is referred to [21]. For class NRL it is referred to [25].

Before defining a validation schema, it is important to briefly discuss the interleaving of phases. During the validation of one transaction, a consistent view on the data is required. If during the validation the data changes, validation might be wrong and as a result inconsistencies would arise. Notice, if validations succeeds the data will be written. Therefore, it is usually not allowed that writing and validating data

⁴CSR means, two operations op_i of ta_i and op_j of ta_j conflict on the same data item if one of them is a write operation. If there is any cycle in the conflict graph, serializability is no longer possible. Bear in mind that serializability testing is NP complete [4].

runs concurrently. So a transaction entering its validation phase requires exclusive access to its WS . An algorithm for validation is sketched out below (Figure 4). Notice, this is just a possible algorithm and one variation is to let a transaction wait and not abort in case of an intersection. A brief discussion about OCC and validation is provided in Section IV-A6.

Let

- 1) dc be a disconnected component
- 2) T_i be the set of transactions to read the data and T_j to write the data.
- 3) $RS_i \subseteq R \cup RC \cup NRO$ be the set of data that is read by T_i
- 4) $WS_j \subseteq R \cup RC \cup NRO$ be the set of modifications.

If T_j enters the validation it has to perform the following test:

```

 $\forall T_k \mid T_k$  is in its validation or write phase:
  if( $RS_i \cap WS_k = \emptyset \wedge WS_j \cap WS_k = \emptyset$ )
    abort dc
  else
     $\forall T_k \mid T_k$  has committed already:
      if( $RS_i \cap WS_k = \emptyset \wedge$ 
          $WS_j \cap WS_k = \emptyset \wedge$  no constraint violation)
        write
      else if( $RS_i \cap WS_k = \emptyset \wedge$ 
               $WS_j \cap WS_k \neq \emptyset \wedge$  no constraint violation  $\wedge$ 
               $WS_j \subseteq R \cup RC$ )
        reconcile
      else
        abort
      end if
    end for
  end if
end for

```

Fig. 4: Algorithm for validation

Owing to the recursive nature of our model, consistency of a composite TA is only successful if all children pass validation.

F. Isolation

Since transactions are composed together and the model allows for partial commits of a T if a compensation exists, other transactions may read a state that is invalidated by a compensation later. Such a situation can lead to a cascading behaviour of compensation or even worse, inconsistencies can result. In advanced transaction management, the concepts of closeness and its opposite openness describe the visibility of results. If a transaction's result is passed to its parent only a "Closed Nested Transaction" [26] is given, if also siblings, or even all unrelated transactions are allowed to read results, if the global outcome has not been determined yet an "Open Nested Transaction" [20], [27] is given. Both the open and closed transaction model have been subject of considerable research (see Section IV).

TABLE III: Data classes (“Reconciliation” refers to [6], “Escrow” to [21].)

	Class	Condition	recommended CC mechanism
1	Reconcilable <i>R</i>	$dep \wedge in \wedge com \wedge \neg cons \wedge num$	Reconciliation
2	Reconcilable with constraint <i>RC</i>	$dep \wedge in \wedge com \wedge cons \wedge num$	$P(X) < th$: Reconciliation, $P(X) \geq th$: Escrow
	Non-reconcilable <i>NR</i>	$(\neg dep \vee \neg in)$	
3	Non-reconcilable Escrow <i>NRE</i>	$(\neg dep \vee \neg in) \wedge com \vee cons \wedge num$	Escrow
4	Non-reconcilable OCC <i>NRO</i>	$(\neg dep \vee \neg in) \wedge (\neg num \vee \neg com) \vee cons$	OCC
5	Non-reconcilable Lock <i>NRL</i>	all <i>NR</i> where a lock is semantically required (justified)	Strict 2PL

The focus in the following paragraphs is on the relationship between the data classes and openness and closeness, respectively. We believe it would be beneficial to exploit the information given by a data class and to determine to use either a closed or an open isolation for a transaction. Usually, this is up to the application developer and may lead to unexpected inconsistencies. With this information in hand, however, a transaction manager could provide support accordingly. Another focus is to incorporate the results of [1] into this work.

For *R* and *RC* data, it is possible to use an open isolation. The state of data at read time does not affect the commit, thus, neither rollback nor a compensation affects the commit. Hence, for data classes *R* and *RC* isolation is not a real concern and transactions operating on *R* or *RC* data only, can release their results immediately and choose an open isolation. The compensation for reconciliation is defined by the inverse of its dependency function.

For escrow data *NRE*, it is irrelevant for other transactions if a transaction rolls back or compensates. To roll back means to recall the granted guarantee and the worst thing that could happen is that another transaction is not able to get a guarantee because this recalled guarantee denied a guarantee to another transaction. An inverse operation is also determinable for escrow data. Hence, transactions operating on *NRE* data only can release their results immediately and use an open isolation.

For *NRO* data, the situation is difficult and the isolation depends on the use case. Moreover, the conflict rate of an entity may be different depending on the time or location. And of course, if no compensation is definable a closed model is required.

For *NRL* data, where locking is justified due to functional requirements, a closed model should be used. Locking in this model is justified to prevent other transactions from reading until the transaction has terminated. So, pending results should also be protected. Assumed *NRL* data is pivotal for inconsistencies or cascading compensation, the requirements based justification to use locks is rather questionable. Recall, the motivation for locks in the above discussion was to support an owner role.

The next step is to incorporate the findings of [1] concerning openness and closeness into this work.

In [1], a closed nested transaction is used for all transactions that are executed against the database layer. Hence also for data of class *R*, *RC*, and *NRE* (depending on the use

case for *NRO* possibly too), which is according to the last deliberations not required and a mixed isolation could be applied instead. For example, a transaction executes one transaction t_{NRE} to write the booking of a flight where entity A represents the available seats and is classified as *NRE*. And, another t_{NRO} to add the actual booking reflected by an entity B classified as *NRO*. Mixed isolation thereby means that t_{NRE} can unilaterally commit and in case of an abort needs to run a compensation. Transaction t_{NRO} , however, needs to await the global outcome to finally commit.

For the composition of software components, [1] suggests an open model. This complies with standards like the Business Activity protocol [28] designed for workflow support and specifically based on the ideas of the open nested transaction model. The composition of software components to construct new applications requires flexible transactional support and must cope with a long living nature and hence the compensation of transactions. This model adopts a utilisation of the open model at the application level. In case a disconnected component defines an open isolation but processes *NRL* data (or *NRO* with a high potential for conflicts), the transaction manager is able to take action and could either set the isolation to close automatically or just inform the developer about the potential risk.

Eventually, we adopt the findings of [1] with the exception to allow for a mixed isolation for composite transactions whose children run against the database layer.

IV. LITERATURE REVIEW AND RELATED WORK

Due to the amount of work that has been carried out in transaction management, a rather large amount of existing work relates to ours.

A. Literature Review

1) Nested Transaction Models: The “Nested Transaction Model” [26] was an influential extension of the flat transaction model and a transaction is modelled as a set of recursively defined sub-transactions resulting in a tree of transactions, where leafs are flat transactions representing data operations. In the nested model a child transaction is only allowed to start when the parent has started and a parent in turn can only terminate if all its children have terminated. If a child fails the parent can initiate alternatives, a so called contingency sub-transaction. However, if the parent transaction aborts all its

children are obligated to also abort. This in turn requires to rollback already committed results.

An extension to the nested transaction model was developed by Weikum and Schenk [27] who introduced the “Open Nested Transaction Model” that allowed, in contrast to Moss’s model, other concurrent sub-transactions to read pending results. In Moss’s model only a parent is allowed to read the result of its children, however, in the open version other concurrent sub-transactions are also allowed to read committed results. The results of a child transaction are only durable after the commit of the parent. To prevent inconsistencies only those sub-transactions that commute with the committed ones are allowed to read their results. Read transactions commute for instance.

From an implementation point of view, the nested transaction model can be emulated by savepoints, furthermore the model is a generalisation of savepoints (see [25], Chapter 4.7). Gray and Reuter also discuss the distinction between nested and distributed transactions and one difference is the nested structure is determined by the functional decomposition of the application, hence how the application views a “Sphere of Control” [18]. The structure of a distributed transaction is determined by the distribution of the data. For example, if a transaction must join two tables each stored at a different node, then of course, the transaction must access both nodes and can be modelled and executed as a nested transaction, but the dependencies are different. In the open variant nested model a transaction can commit and abort independently, whereas a commit or abort in the distributed model always depends on each other and only one outcome is possible.

The closed and open nested transaction model can be seen as the seminal work concerning Advanced, Workflow, or Business Transaction Models (see [7], [16], [17]).

An important transaction model that provides a formal correctness criteria for compensation is the Sagas model [12]. A Saga is a transaction that consists of a set of ACID sub-transactions, however with a predefined execution order, and further a set of corresponding compensating sub-transactions must be defined. Notice a compensation transaction is mandatory for each sub-transaction. A Saga completes successfully if either each sub-transaction has committed or the corresponding compensation sub-transaction commits. Generally, a Saga transaction differs from a Chained Transaction [25], where already committed results cannot be undone, but especially this is important to fit the requirements for long-lived transactions. Moreover, a compensation transaction allows for a relaxation of full isolation, also atomicity, and increases inter-transaction concurrency. Locks can be released as soon as a (sub-)transaction commits, even if the parent is still active because the rollback is performed by a separate compensation transaction which does not affect the serializability. And, since each transaction in the Sagas model must define a compensation transaction, also the cascading of errors can be handled, at least theoretically. An extension of the Sagas model is the “Nested Sagas” model [29] that provides mechanisms to structure steps involved within a long running transaction into hierarchical

transaction structures. However, one drawback of each model that heavily applies compensation is the requirement for a compensation operation and as soon as non compensatable transactions exist, compensation is not feasible.

In this section, some important aspects of nested transactions and the compensation have been briefly explained. From a historical view, nested transaction models and the so called “Advanced Transaction Models” (ATM) [16] were the foundation for a new generation of transaction models, so-called workflow transaction models [30]. Advanced Transaction Model are sometimes claimed to be less general and more application specific compared to the ACID model. Workflow transaction models do not meet this criticism to the same extent.

Compared to a nested transaction where usually only leaf nodes represent data operations, within a workflow transaction model each node can modify data. A more general definition is: the flat transaction model has evolved vertically to transaction trees, whereas workflows or generally long-computations represent also a horizontally evolution. Both workflow transaction models and ATM have been rarely implemented in the database layer, rather they have been applied in transaction coordination protocols at the middleware level.

2) *ConTracts*: Another important transaction model for long-computations is the ConTracts model, introduced by [31] and revised in [32]. This is a *conceptual framework for the reliable execution of long-lived computations in a distributed environment*. The core module of the ConTracts model is the ConTract script that describes a long-lived computation similar to a workflow model. The steps involved within this computation are not single statements but represent programs, methods, or applications, which can be invoked through a call interface. In the ConTracts model, the application is responsible for what happens inside a step, and the ConTract script is responsible for keeping the control flow alive between the involved steps, i.e., applications. Similar to the Saga model, each step must define its compensation step to relax isolation, reduce blocking time and thereby increase parallelism. In addition, each step must define an pre- and post-condition (the ConTracts model calls pre- and post-condition entry- and exit-invariant. So, the scope of an in-variance is the step. However, we believe that these invariants are actually conditions as the scope of an invariant should be the governing application. That is, an invariant needs to be true over the entire computation). The pre-condition must hold (validate to true) before the step can be invoked. For instance, an pre-condition can check if the data required by the step is locked. So while compensation facilitates non-blocking, the pre-condition can ensure non-blocking. Beside the pre-condition an post-condition must also validate to true before control can pass to the next step. The concept of post-conditions allows other steps executed in the future to step back, thus an post-condition can be part of another pre-condition. The ConTracts model provides its own definitions of transactional properties and recovery, and to avoid the shortcomings of the atomicity property a two-layered recovery mechanism has been introduced. Recovery

and Serializability are derived from the classic serializability and strictness and the notion of “Prefix Reducibility” [4]. The difference, however, is, since ConTracts are not isolated, the model introduces conditions that define the structural dependencies among the different steps. Such a definition however is not as straightforward as for conflicts between write operations within a schedule. Not only the data an operation accesses must be considered, but the semantics of the whole step must be considered to determine any conflict with other activities.

Regarding recovery, the model distinguishes between recovery at the step and at the script level. Recovery at the script level is important to keep the overall computation consistent, which means, after a failure of a step occurs other steps active during this time need to be recovered, too; this so called forward recovery is compensation.

One strength of the ConContract model is its precise definitions for compensation. The general problem of compensation is that if compensation of a step is required no other concurrent transaction should violate the compensation itself. This leads to the definition of “indirect compensatability”, “indirect compensation chain”, and “absolute compensatability”. Indirect compensatability defines that within an ordered execution of two steps s_1 and s_2 with their corresponding compensation steps cs_1 and cs_2 the compensation must follow the same order to maintain the consistency. The indirect compensation chain then consists of “*all steps for which the indirect compensation relation holds*.”. Absolute compensatability defines a compensation step that is independent of any other compensation step, thus, even an arbitrary execution leads to a compensated result. The ConTracts model makes no further statements how far a running workflow must be rolled back. Based on the definitions provided within the model a complete roll back is foreseen. The partial rollback of workflows is particularly addressed in Leymann’s concept of “spheres of joint compensation” [19].

Leymann’s work [19] basically is built upon the concept of spheres of control, Sagas, and the ConContract model. The novelty of his concept is to explicitly enable a partial rollback of an active workflow and not to rollback the entire workflow, or more generally: the entire affected tree of transactions. This reflects more the real situation of long-lived computations in which not all work must be undone.

The idea is to define a sphere as: “*any collection of activities (steps) of a process is called Sphere of Joint Compensation (sphere) if either all activities have run syntactically successful or all activities must have compensated*” [19]. The compensation itself is modelled by adding a compensation activity to each sphere and activity.

The compensation itself is basically performed by the execution of all compensation steps of each activity in reverse order. A composed activity, that is an activity consisting of other activities, can request a shallow or a deep compensation and in the latter case all associated compensation operations of the composed activities have to be executed too. If a shallow compensation is requested only the compensation associated

with the root activity is executed. An integral compensation only performs the compensation associated with the sphere and running the compensation of each encompassed activity is referred to as discrete compensation. Leymann also provides compensation modes that define how compensation can impact the neighbourhood of the activity or sphere, the so called “Proliferation Property”. When an activity must be compensated the proliferation property defines if the compensation of the whole sphere must be executed, too. This, in turn, must validate the integral property and either the sphere’s compensation must be executed only, or the compensation of each activity. Since spheres can overlap, the model provides a definition of how to treat cascading compensation. Generally, this compensation model can be seen as very complete beside the general drawbacks of compensation discussed in the next section, and its concepts have been considered by the Business Process Execution Language (BPEL) and Business Process Modelling Notation (BPMN) standard.

3) Concurrency Control for Transactional Processes: The theoretical framework by Schuldt et al. [33] to reason about concurrency control and recovery in transaction processes is an attempt to unify the theory of concurrency control and recovery for transactional business processes (processes) or workflows. Schuldt et al. argue that the challenge we face is to design a single correctness criterion for both concurrency control and recovery that also copes with the added structure found in processes. They further observed that the flow of control introduced by processes is one of the basic semantic elements, and that a correct execution must obey the already existing ordering constraints among their different operations and alternative executions. These constraints determine how activities of the process can be interleaved during execution. They further state that the different atomicity properties among the involved systems can not always fit the strong requirements of models applying compensation, e.g. ConTracts, where each operation requires its inverse because it is not guaranteed to find an inverse.

Similar to other models, they extend the notion of atomicity by considering also alternatives or a partial rollback of already executed steps within the process. In practice, tasks are often executed in parallel to increase the time to market, and regarding concurrency control without considering recovery an ordering of these tasks is sufficient, however if recovery is taken into account, and for one of the steps no compensation exists, the situation becomes different. Their example is a production and a corresponding test within a manufacture where the production usually has no inverse function (at least no acceptable one). Thus, the production is only allowed to start if the test terminated successfully because a concurrent execution can lead, in the case of compensating the test, to an invalid production if both are executed concurrently.

The general point they address is more how to maintain correctness if no compensation is given. What follows in their paper is a theoretical model for correct process schedulers basically oriented on the Flex Model [34], [35]. The Flex model introduces, beside the notion of a compensatable sub-

transaction, a “retriable” sub-transaction that can be retried and eventually succeeds if retried a sufficient number of times, and a “pivot” sub-transaction that is neither retriable nor compensatable. Concerning the details of the framework and its definition for correct schedules, a more general explanation is provided here. At the database level the serializability of operations can be expressed by their conflict relationship, and similar a conflict relationship is defined between activities, however, at the process level. But, since the internals of an operation are usually not exposed by an activity the whole activity needs to be classified to define its conflict relationship to other activities. Based on the classes of retriable, compensatable and pivot sub-transactions, activities are classified and a commutativity, a compensation and a effect-free activity rule can be expressed. These rules in turn can be exploited by a process task scheduler to produce a correct process schedule. Summarising, their framework is based on the Flex transaction model and provides the theoretical foundation to ensure and reason about the correctness of process schedules by considering both, concurrency control and recovery within one model.

4) Web Services and Transactions: A model for the distributed management of concurrent Web service transactions is introduced in [36]. Alrifai et al. claim that in the “*open and dynamic Web service environment, business transactions enter and exit the system independently and under relaxed isolation transactional dependencies can emerge among independent business processes which must be taken into account when compensation is required in order to avoid inconsistency problems.*” Within a closed environment inconsistencies can be controlled more easily because the dependent transactions are known. Their work combines an optimistic decentralised variant of the SGT (Serialization Graph Testing) protocol that applies an Edge Chasing algorithm to detect potential global waiting cycles with a transaction scheduling algorithm that selects the service provider based on their scheduling offers. The architecture foresees a multi-layered architecture consisting of a process, a Web service and a resource level and applies the Multilevel Nested Transaction Model [4], [27] where each leaf has the same distance to the root. Conflicts at the resource level are detected by a separate resource-level concurrency control module, at the service level a service-level concurrency control module is responsible to detect conflicts. Conflicts are usually defined by conflict relations expressed as a transactional dependency graph in their model and global consistency is ensured if each local system guarantees local consistency as in distributed database systems (see commit-order and rigorous scheduling [4]). They apply a 2PC as atomic commitment protocol.

Their model extends the OASIS Web-Service standards WS-Coordination [37], WS-Atomic Transaction [38], and WS-Business Activity [39] (see section IV-A5). However, their Multilevel Nested Transaction Model only allows commutative concurrent sub-transactions to read pending results. To define the commutativity of transactions each transaction type is divided into atomic steps with compatibility sets according

to its semantics. Transaction types that are incompatible and are not allowed to interleave at all. Farrag and Özsü [40] already refined this method by allowing some interleaving for incompatible types and assuming fewer restrictions for compatibility. The problem is that finding the compatibility sets for each transaction step is a $\mathcal{O}(n^2)$ problem. Alrifai et al. apply a conflict matrix that defines the conflict sets of a transaction and hence their approach must deal with a quadratic complexity too, even their conflict predicates do not solve the problem because their purpose is only to enable a conflict detection across autonomous and independent systems.

5) Web Service Transaction Standards: This section briefly explains and mentions some specification for protocols that allow for distributed transaction processing in a XML Web service (WS) architecture. The reason for such a brief explanation is that the specifications are based on the ideas of the nested and long running transactions, which have been discussed already (see section IV-A1).

The so called “WS Transactions specifications” are the “WS Coordination” [37], the “WS Atomic Transaction” [38], and the “WS Business Activity” [39] specification. In terms of a WS architecture, they relate to the Quality of Service (QoS) layer and all these specifications are built on top of the Simple Object Access Protocol (SOAP) and Web Service Description Language (WSDL) standards.

The core element is the WS-Coordination (WS-CO) framework that is an “*extensible framework for providing protocols that coordinate the actions of distributed transactions*” [37]. The framework allows for a mechanism to register partners and to allow for a generic control of their interaction. If a transactional conversation is required an additional framework that provides a specific atomic commitment protocol, a so called completion protocol, must be plugged into the WS-CO framework. The WS Atomic Transaction or the WS Business Activity framework are such frameworks that provide completion protocols.

The WS Atomic Transaction (WS-AT) framework provides two completion protocols namely a volatile 2PC and a durable 2PC. Whereas the first one is intended for services that operate on volatile, i.e., non persistent data, the second one is, as the name suggests, for services operating on persistent data. Both protocols can be used within the same transaction. However, in such a case the volatile services must complete before the durable ones. Beside the extensions required to comply with the WS specification family, the WS-AT specification addresses the well-known 2PC as introduced by “The Open Group” [41] and allows for distributed “all-or-nothing” transactions.

In contrast to the all-or-nothing principle of WS-AT, WS Business Activity (WS-BA) allows for hierarchical nested scopes possibly requiring compensation, relaxed isolation, autonomous participants, or abort autonomy for instance. Generally, WS-BA is a specification for the management of nested transaction and a so called “mixed outcome” of a transaction is possible; for example some transactions terminate committed, some aborted, and others compensated.

In summary, the WS transaction specifications, especially the WS-BA is an example for a technology that is able to cope with nested and complex transactions and for more details on WS-CO, WS-AT or the WS-BA it is referred to the specifications.

6) *Optimistic Concurrency Control*: In the early eighties, Kung [13] introduced OCC, which has not gained as much consideration as CC protocol in commercial database systems as locking has. The PyrrhoDB [42] is one database we are aware of that implements OCC as CC mechanism.

Härder [14] sees the challenge in merging the workspace of a transaction with the actual state of the database. “*Their essential problem consists of merging these copies during COMMIT processing thereby regaining a transaction-consistent database image*”. This asynchronism is especially challenging “*when these copies do not match with the units of transfer (pages)*” [14] and when different copies of the database have to be kept consistent. Notice, Härder’s critique about the merging of copies seems obsolete and outdated as Multi-Version Concurrency Control protocols as widely used by Oracle or PostgreSQL, for example, are subject to the same problem of merging copies.

Härder goes even further and states that even if OCC has been defined for applications where conflicts are unlikely, “*locking also behaves quite well in such a particular environment (no wait or deadlock conflicts), there seems to be little reason to introduce a specialised control mechanism*”. Härder refers to an empirical comparison that shows that with OCC the abort rate of transactions is higher compared to locking. This is due to the property that in locking a transaction waits rather than restarts. In this case the restart of a transaction must be considered as being equivalent to an abort. In other words, as stated by Härder, to wait increases the probability for success. It is worthwhile to discuss this issue in a little bit more detail here. One particular problem with OCC is that it is not possible to get any guarantees in advance. “First come, first served” is the result of the optimistic behaviour. By contrast, in locking a sophisticated locking schema can guarantee a transaction that updates will succeed if constraints are not violated, and in the absence of physical errors. Hence, if the update rate on a field is high, the abort rate of locking is less than with OCC, but for the price that a transaction has to wait. In this sense waiting is not wasting.

Franaszek et al. [23] come up with the idea to investigate “*the potential reduction in the required concurrency level via the use of what might be viewed as the pre-fetch property of transactions which run but do not commit*.” These transactions are said to run in a “virtual execution mode”. Particularly this means a transaction is executed twice. The first execution is to determine the required data, calculate the access paths, and load the data into the cache. The second execution is to actually commit the transaction with all the data in cache already and a lock pre-climbing that requests all locks in one atomic step, which is feasible since the entire data set is known already. So, the idea of Franaszek’s et al.’s mechanism is not to restart a transaction even though it is known that the

transaction will fail at commit time and to benefit during the second execution from the information gathered at the first run instead. One of the most important considerations thereby is the notion of “Access Invariance” that is a transaction will “*with high probability perform the same operations on the same subset of objects without regard to the implied serial order of execution*.” This assumption is adequate as it would mean the correctness notion of serializability would be inadequate else. The authors emphasise that there can be indeed conflicts between transactions or constraints that might permit transactions from commit, but neither of them does affect the access invariance in general.

Despite the rather sceptical note by Härder as well as by Mohan [43] and the fact that OCC has not been implemented by many commercial database vendors as a basic CC mechanism, OCC fits well into a disconnected computing architecture where consistency preserving mechanisms are required as part of a Middleware solutions (see [44]) Laux et al. [45] thoroughly analyse Row Version Verification (RVV) as an implementation of OCC if the database does not support “optimistic locking” per se and their patterns to implement RVV for some common databases and data access technologies at the MW layer fits well into a disconnected architecture.

B. Related Work

As presented in the last section, there are many transaction models that consider a nested and recursive transaction structure. The ACTA framework [46], [47] provides an independent language to describe these complex transaction models by demarcating aspects of atomicity and isolation. Its drawback is the large terminology and the missing execution semantics. Its nature is purely descriptive. Eventually, beside the aforementioned transaction models and their well understood concepts, the ACTA framework also inspired our work. However, we believe that to use Boolean expressions to describe transactional dependencies is easier to comprehend and reduces the large terminology, which can be found in ACTA. And, a satisfaction expressions is computable.

The second key piece of this work, the classification of data, is inspired by [3] and [6]. Kraska et al. use statistical measures to determine a conflict probability and adapt the CC mechanism accordingly. Since their work focuses on the Cloud, replication plays an important role too in their model. By contrast, this work does not consider replication. As described in Section III-D, the data classes are according to Laux and Lessner’s work on Reconciliation [6] and O’Neil’s work [21] on the Escrow data type. Furthermore, we are specifically interested in the question how the classification of data can be exploited also concerning the isolation property.

Many domains, for example, object orientation, SOC, or Mobile Computing are confronted with disconnected situations if an increased local autonomy is required. The assumption in this work, that there is actually no difference, and all software components should just run in a disconnected mode with separate read and write phase. This is inspired by work on OCC [13], [14], [22], [23], [45] and to divide a

component into such phases is also similar to the Command Query Pattern [10], however, in a larger scale.

V. CONCLUSION, CONTRIBUTION AND FUTURE WORK

The contribution of this work is an expression language for transactional dependencies if transactions are composed together. An expression is based on Boolean algebra and can be used by a transaction manager to coordinate the transactional composition because the execution semantics is derivable.

Moreover, since we believe that an application of a CC mechanism should consider the semantics of data and operations to better trade-off the consistency needs and incurring costs, and not follow the one CC mechanism fits all needs paradigm, we have introduced five different data classes using a certain CC mechanism. In this context we have also analysed the implications for isolation in a nested transaction. Moreover, our classification complies with an optimistic attitude, which in turn complies with the needs of current disconnected and asynchronous computing architectures.

To impose a phase structure on disconnected components and analyse the implications on transaction management is another contribution.

Our goal in the long term is to allow for even better trade-offs similar to [3]. In our vision applications should express their transactional requirements like the required level of consistency, processing time, and costs. Due to the composition this also requires a model that copes with the composition of requirements, raising the needs for metrics. The classification of data and a language to express transactional dependencies are first steps. For the future, we also envision to run transactions in different lanes according to their semantics and requirements. Such an allocation and division could help to easier verify and trade-off scalability and consistency demands.

Simulation results that justify a classification of data are probably the most missing piece in this work. Currently, a prototypical transaction simulation and reasoning framework is still under development.

REFERENCES

- [1] Tim Lessner, Fritz Laux, Thomas Connolly, Cherif Branki, Malcolm Crowe, and Martti Laiho. An optimistic transaction model for a disconnected integration architecture. In *DBKDA 2011, The Third International Conference on Advances in Databases, Knowledge, and Data Applications*, pages 186 – 191, 2011.
- [2] Jim Gray and Pat Helland and Patrick E. O’Neil and Dennis Shasha. The Dangers of Replication and a Solution. In H. V. Jagadish and Inderpal Singh Mumick, editors, *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, Montreal, Quebec, Canada, June 4-6, 1996*, pages 173–182. ACM Press, 1996.
- [3] Tim Kraska, Martin Hentschel, Gustavo Alonso, and Donald Kossmann. Consistency rationing in the cloud: pay only when it matters. *Proc. VLDB Endow.*, 2:253–264, August 2009.
- [4] Gerhard Weikum and Gottfried Vossen. *Transactional Information Systems: Theory, Algorithms, and the Practice of Concurrency Control and Recovery*. Morgan Kaufmann, 2002.
- [5] Hector Garcia-Molina, Jeffrey D. Ullman, and Jennifer Widom. *Database systems - the complete book (2. ed.)*. Pearson Education, 2009.
- [6] Fritz Laux and Tim Lessner. Escrow Serializability and Reconciliation in Mobile Computing using Semantic Properties. *International Journal On Advances in Telecommunications*, 2(2):72–87, 2009.
- [7] Ahmed Elmagarmid, Marek Rusinkiewicz, and Amit Sheth, editors. *Management of heterogeneous and autonomous database systems*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999.
- [8] Margaret H. Dunham, Abdelsalam Helal, and Santosh Balakrishnan. A mobile transaction model that captures both the data and movement behavior. *Mob. Netw. Appl.*, 2(2):149–162, 1997.
- [9] Thomas Erl. *SOA Design Patterns*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 2009.
- [10] Bertrand Meyer. *Object-Oriented Software Construction*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1st edition, 1988.
- [11] Michael Beisiegel et al. Service component architecture (sca) v1.00, 2010-11-08, 2007.
- [12] Hector Garcia-Molina and Kenneth Salem. Sagas. In *SIGMOD ’87: Proceedings of the 1987 ACM SIGMOD international conference on Management of data*, pages 249–259. ACM, 1987.
- [13] H. T. Kung and John T. Robinson. On optimistic methods for concurrency control. *ACM Trans. Database Syst.*, 6(2):213–226, 1981.
- [14] Theo Härdler. Observations on optimistic concurrency control schemes. *Inf. Syst.*, 9:111–120, November 1984.
- [15] G. D. Walborn and P. K. Chrysanthis. Supporting semantics-based transaction processing in mobile database applications. In *SRDS ’95: Proceedings of the 14TH Symposium on Reliable Distributed Systems*, page 31. IEEE Computer Society, 1995.
- [16] Ahmed K. Elmagarmid, editor. *Database transaction models for advanced applications*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1992.
- [17] Sushil Jajodia and Larry Kerschberg, editors. *Advanced Transaction Models and Architectures*. Kluwer, 1997.
- [18] C. T. Davies. Data processing spheres of control. *IBM Systems Journal*, 17(2):179–198, 1978.
- [19] Frank Leymann. Supporting Business Transactions Via Partial Backward Recovery In Workflow Management Systems. In *BTW*, pages 51–70, 1995.
- [20] Alejandro Buchmann, M. Tamer Özsu, Mark Hornick, Dimitrios Georgakopoulos, and Frank A. Manola. *A transaction model for active distributed object systems*, pages 123–158. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1992.
- [21] Patrick E. O’Neil. The escrow transactional method. *ACM Transactions On Database Systems*, 11:405–430, December 1986.
- [22] P.A. Franaszek, J.T. Robinson, and A. Thomasian. Access invariance and its use in high contention environments. In *Data Engineering, 1990. Proceedings. Sixth International Conference on*, pages 47 –55, February 1990.
- [23] A. Thomasian. Distributed optimistic concurrency control methods for high-performance transaction processing. *Knowledge and Data Engineering, IEEE Transactions on*, 10(1):173 –189, jan/feb 1998.
- [24] Alexander Thomson and Daniel J. Abadi. The case for determinism in database systems. *Proc. VLDB Endow.*, 3:70–80, September 2010.
- [25] Jim Gray and Andreas Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, 1993.
- [26] J. Eliot B. Moss. *Nested transactions: an approach to reliable distributed computing*. Massachusetts Institute of Technology, Cambridge, MA, USA, 1985.
- [27] Gerhard Weikum and Hans-Jörg Schek. Concepts and Applications of Multilevel Transactions and Open Nested Transactions. In *Database Transaction Models for Advanced Applications*, pages 515–553. Morgan Kaufmann, 1992.
- [28] Web services business activity (ws-businessactivity) version 1.1. Online (accessed 15.08.2011), July 2007.
- [29] Hector Garcia-Molina, Dieter Gawlick, Johannes Klein, Karl Kleissner, and Kenneth Salem. Modeling long-running activities as nested sagas. *Data Eng.*, 14(1):14–18, 1991.
- [30] Ting Wang, Jochum Vonk, Benedikt Kratz, and Paul Grefen. A survey on the history of transaction management: from flat to grid transactions. *Distrib. Parallel Databases*, 23(3):235–270, 2008.
- [31] Helmut Wächter and Andreas Reuter. The ConTract Model. In *Database Transaction Models for Advanced Applications*, pages 219–263. Morgan Kaufmann, 1992.

- [32] Andreas Reuter and Kerstin Schneider and Friedemann Schwenkreis. ConTracts Revisited. In Sushil Jajodia and Larry Kerschberg, editors, *Advanced Transaction Models and Architectures*. 1997.
- [33] Heiko Schuldt and Gustavo Alonso and Hans-Jörg Schek. Concurrency Control and Recovery in Transactional Process Management. In *Proceedings of the Eighteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 31 - June 2, 1999, Philadelphia, Pennsylvania*, pages 316–326. ACM Press, 1999.
- [34] Sharad Mehrotra and Rajeev Rastogi and Henry F. Korth and Abraham Silberschatz. A Transaction Model for Multidatabase Systems. In *ICDCS*, pages 56–63, 1992.
- [35] Ahmed K. Elmagarmid and Yungho Leu and Witold Litwin and Marek Rusinkiewicz. A Multidatabase Transaction Model for InterBase. In Dennis McLeod and Ron Sacks-Davis and Hans-Jörg Schek, editors, *16th International Conference on Very Large Data Bases, August 13-16, 1990, Brisbane, Queensland, Australia, Proceedings*, pages 507–518. Morgan Kaufmann, 1990.
- [36] Mohammad Alrifai and Peter Dolog and Wolf-Tilo Balke and Wolfgang Nejdl. Distributed Management of Concurrent Web Service Transactions. *IEEE T. Services Computing*, 2(4):289–302, 2009.
- [37] Oasis. Web Services Coordination (WS-Coordination), 2009.
- [38] Oasis. OASIS Web Services Atomic Transaction Version 1.2, 2009.
- [39] Oasis. OASIS Web Services Business Activity Version 1.2, 2009.
- [40] Abdel Aziz Farrag and M. Tamer Özsu. Using semantic knowledge of transactions to increase concurrency. *ACM Trans. Database Syst.*, 14(4):503–525, 1989.
- [41] The Open Group. Distributed transaction processing: The XA specification, 1992.
- [42] Malcolm Crowe (University of the West of Scotland). The Pyrrho database management system (<http://www.pyrrhodb.com/>, 2010-11-02), 2010.
- [43] Mohan. Less optimism about optimistic concurrency control. In *Proceedings of the 2nd International Workshop On Research Issues In Data Engineering*, pages 199–204, 1992.
- [44] Philip Bernstein and Eric Newcomer. *Principles of transaction processing: for the systems professional*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2009.
- [45] Fritz Laux, Martti Laiho, and Tim Lessner. Implementing row version verification for persistence middleware using sql access patterns. *International Journal on Advances in Software*, issn 1942-2628, 3(3 & 4):407 – 423, 2010.
- [46] Panayiotis K. Chrysanthis and Krithi Ramamritham. ACTA: a framework for specifying and reasoning about transaction structure and behavior. *SIGMOD Rec.*, 19(2):194–203, 1990.
- [47] Panos K. Chrysanthis and Krithi Ramamritham. Synthesis of extended transaction models using acta. *ACM Trans. Database Syst.*, 19:450–491, September 1994.

Verifiable Constraints for Ambients of Persistent Objects

Suad Alagić and Harika Anumula

Department of Computer Science

University of Southern Maine

Portland, Maine, USA

alagic@usm.maine.edu, harika.anumula@maine.edu

Akinori Yonezawa

Advanced Institute of Computational Science

Kobe, Japan

yonezawa@riekn.jp

Abstract—This paper develops a typed object-oriented paradigm equipped with message-based orthogonal persistence. Messages in this paradigm are viewed as typed objects. This view leads to a hierarchy of types of messages that belong to the core of typed reflective capabilities. Unlike most persistent object-oriented models, this model is equipped with general integrity constraints that also appear as a hierarchy of types in the reflective core. A transaction is naturally viewed as a sequence of messages and it is equipped with a precondition and a postcondition. The presented framework is motivated by ambients of persistent concurrent and mobile objects. The practical result supporting the developed model is a verification technology for ambients of persistent objects based on a higher-order verification system. This technology applies to static interactive verification of transactions with respect to the schema integrity constraints.

Keywords-Object databases; constraints; reflection; transactions; verification.

I. INTRODUCTION

The current object technology has nontrivial problems in specifying classical database integrity constraints, such as keys and referential integrity [11][14][15]. No industrial database technology allows object-oriented schemas equipped with general integrity constraints. In addition to keys and referential integrity, such constraints include ranges of values or number of occurrences, ordering, and the integrity requirements for complex objects obtained by aggregation [2]. More general constraints that are not necessarily classical database constraints come from complex application environments and they are often critical for correct functioning of those applications [3].

Object-oriented schemas are generally missing database integrity constraints because those are not expressible in type systems of mainstream object-oriented programming languages. Since the integrity constraints cannot be specified in a declarative fashion, the only option is to enforce them procedurally with nontrivial implications on efficiency and reliability. The constraints must fit into type systems of object-oriented languages and they should be integrated with reflective capabilities of those languages [18]. Most importantly, all of the above is not sufficient if there is no technology to enforce the constraints, preferably statically,

so that expensive recovery procedure will not be required when a transaction violates the constraints at run-time [2][3].

The object-oriented database model presented in this paper integrates message-based orthogonal persistence, object-oriented schemas equipped with general integrity constraints accessible by reflection, and transactions that are required to satisfy the schema integrity constraints. The model is based on a type system and it offers a significantly different view of messages in comparison with the mainstream object-oriented languages. The model applies to ambients of persistent and concurrent objects.

A message in mainstream object-oriented languages such as Java or C# is specified in a functional notation. This functional view fits messages that cause no side-effects and report the properties of the hidden object state. The functional view also fits queries. Other categories of messages do not fit the functional notation. An update message is a message that changes the state of the receiver and possibly other objects as well. An update message does not have a result and its semantics does not fit the functional notation.

An asynchronous message [23], in general, does not have a result either and hence the functional notation is not appropriate. A particular type of an asynchronous message (a two-way message) has a result, but this result is not necessarily immediately available at the point of the message send. Asynchronous (remote) queries would fit this pattern. A transient message has a limited lifetime and a sustained message does not have this limitation. A message may be one-to-one with a single receiver or a message may be a broadcast message sent to a set of receiver objects. Many messages naturally combine the features of the above mentioned message types. For example, a two-way transient message, a one-to-one query message, a one-to-many sustained update message, etc. [10].

Further development of this approach leads to an orthogonal model of persistence [6] that is based on a special message type that promotes the receiver object to persistence. A transaction is defined as a sequence of messages of different types. Concurrency control and recovery protocols can now be implemented in the object-oriented style. Indeed, serialization protocols require knowledge of types of messages (queries versus updates) and impose an appropriate ordering

of conflicting messages. Similar comments apply to recovery protocols that are in our view sequences of do, undo and redo messages.

Object-oriented constraints are a key feature of the presented model. Specifying the behavior of objects of a message type is naturally done using an object-oriented assertion language. Object-oriented assertion languages allow specification of database integrity constraints as class invariants, declarative specification of transactions with pre and post conditions, and queries whose filtering (qualification expression) is specified as an assertion predicate. The assertion languages used to express constraint-related features of the model presented in this paper are JML (Java Modeling Language) [12], and Spec# [13].

A database transaction is accessing a large amount of data. Checking constraints at run-time is often prohibitively expensive, and violation of constraints may require expensive recovery procedures. The idea of static verification of transactions is not new [8][20][21]. However, all the previous attempts failed to produce results at a practical, applicable level. The main idea is that a transaction is statically verified to satisfy the schema integrity constraints so that either no run-time checking or just limited run-time checking of constraints will be required. This means that data integrity will be provably guaranteed with no penalty on efficiency and a significant increase of reliability.

Two critical pieces of the technology that supports the model presented in this paper are: an extended virtual platform for constraint management and verification techniques that apply to constraints. The extended virtual machine integrates constraints into the run-time type system, allows their introspection and enforcement [18]. Verification techniques apply to object-oriented transactions written in Java or C#. The verification technologies are based on PVS (Prototype Verification System) [3], and automatic static techniques of Spec# [2].

PVS is chosen because of its sophisticated type system which includes predicate subtyping and bounded parametric polymorphism. Because of this the PVS type system is a good match for the type systems of mainstream object-oriented programming languages. In addition, PVS has powerful logic capabilities and as a higher-order system it allows embedding of specialized logics suitable for the object-oriented paradigm such as temporal or separation logic.

We first present in Section II a motivating application based on ambients of concurrent and service objects. The fundamentals of the view of messages as typed objects is developed further in Section III, along with the hierarchy of message types. The model of persistence is described in Section IV. Queries and transactions are discussed in Section V. Type safe reflection, which includes run-time representation of types (including message types) and assertions is the subject of Section VI. Finally, in Section VII, we present our technology based on PVS for static verification of

transactions with respect to the schema integrity constraints.

II. MOTIVATING APPLICATION: AMBIENTS OF CONCURRENT OBJECTS

In this introductory section, we describe the environments that lead to the view of messages as typed objects. An ambient [10] is a dynamic collection of service objects. The types of service objects are assumed to be derived from the type `ServiceObject`. This is why the class `Ambient` is parametric and its type parameter has `ServiceObject` as its bound type as follows:

```
abstract class Ambient
    <T extends ServiceObject> { . . . }
```

When a message is sent to an ambient object, one or more service objects is selected depending upon the type of the message, and the message is sent to those service objects. Messages sent to an ambient are in general asynchronous, hence they are of the type `Message`. When such a message object is created, it has its identity, a lifetime, and behaves according to one of the specific subtypes of the type `Message`. For example, a transient message has a limited discovery time and a sustained message does not. Moreover, messages can be sent to message objects. For example, if a message is a two-way message, a message that refers to the future method may be sent to the two-way message object to obtain the result when it becomes available [23].

An ambient has a filter, which selects the relevant service objects that belong to the ambient. This predicate is defined for a specific `Ambient` class, i.e., a class that is obtained from the class `Ambient` by instantiating it with a specific type of service objects. An ambient has a communication range, which determines a collection of service objects that are in the ambient's range. The reach of an ambient object is then the collection of all service objects of the given type that satisfy the filter predicate and are within the communication range of the ambient object.

The class `Ambient` is equipped with a scheduler, which selects the next message for execution according to some strategy. So the `Ambient` class looks like this:

```
abstract class Ambient
    <T extends ServiceObject> {
        abstract boolean filter(T x);
        Set<Message> messages();
        Set<T> communicationRange();
        Set<T> reach();
        invariant (forAll T x)
            (x in this.reach() <=>
                this.filter(x) and x in
                this.communicationRange());
    }
```

An example of a specific ambient class is

```
class StockBroker extends ServiceObject {
    int quote(String stock);
    int responseTime();
```

```

    . . .
class StockBrokerAmbient
    extends Ambient<StockBroker> {
String[] displayStocks(){. . .};
void requestQuote(String stock){. . .};
boolean filter(StockBroker x)
    {return x.responseTime() <=10;};
}
StockBrokerAmbient stockbrokers =
    new StockBrokerAmbient();

```

In a more general concurrent setting [23], a concurrent object is equipped with its own virtual machine. A virtual machine is equipped with a stack, a heap, a queue of messages, and a Program Counter (PC), as shown in Figure 1.

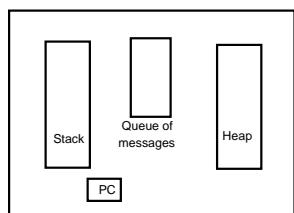


Figure 1. A concurrent object

```

interface ConcurrentObject {. . .}
class ConcurrentObjectClass
    implements ConcurrentObject {
private VirtualMachine VM();
}

```

In a concurrent paradigm of [23], a concurrent object executes messages that it receives by invoking the corresponding methods. In order to be able to do that, the heap of the object's virtual machine must contain reflective classes such as Class, Method, Message, etc. These classes are stored on the heap of the object's virtual machine. The heap also holds the object state. Execution of a method is based on the object's stack according to the standard stack-oriented evaluation model.

A concurrent object gets activated by receiving a message. If a concurrent object is busy executing a method, the incoming message is queued in the message queue of the object's virtual machine. Messages in the queue will be subsequently picked for execution when an object is not busy executing a method. So at any point in time an object is either executing a single message or else it is inactive (i.e., its queue of messages is empty).

In the extreme case, all objects are concurrent objects, i.e., the class `ConcurrentObjectClass` is identified with the class `Object`. A service object is now defined as a concurrent object:

```

interface ServiceObject
    extends ConcurrentObject {. . . }

```

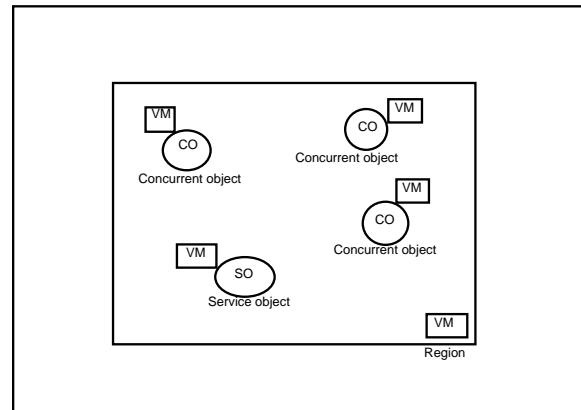


Figure 2. Regions of concurrent and service objects

We can now redefine an ambient in this new setting as a concurrent object, which represents a dynamic collection of concurrent service objects:

```

class Ambient <T extends ServiceObject>
    extends ConcurrentObject {
    . . .
}

```

Since an ambient is a concurrent object, it has its own virtual machine with a queue of messages sent to the ambient object and not serviced yet.

A mobile object is a concurrent object that is equipped with a location:

```

interface MobileObject
    extends ConcurrentObject {
Location loc();
}

```

A region is an ambient that captures the notion of locality. It consists of all concurrent objects within the region as well as the service objects in that region, as illustrated in Figure 2.

```

class Region <T> extends Ambient<T> {
Set<ConcurrentObject> objects();
boolean withinRegion(MobileObject x);
invariant (ForAll MobileObject x)
    (this.withinRegion(x) =>
        x in this.objects());
}

```

For example, if `class Server extends ServiceObject {. . .}` then `Region<Server>` would be an example of a region type. Since a region is a concurrent object, it is equipped with its own virtual machine. Also, since a region is an ambient, it receives messages that are queued in the message queue of the region's virtual machine to be serviced. Servicing a message sent to a region amounts to selecting a server object and sending the message to that server.

III. TYPES OF MESSAGES

Non-functional messages in this paradigm are objects. A message is created dynamically and it has a unique identifier like any other object. In the concurrent architecture described in Section II, object identifiers must be global. The attributes of a message are the receiver object and the array of arguments along with a reference to a method. Messages of specific subtypes will have other attributes. This produces a hierarchy of message types that are subtypes of the type Message.

```
interface Message {
    Method m();
    Object receiver();
    Object[] arguments();
    int timeStamp();
}
```

When a message object is created its time stamp is recorded. The implementing class would have a constructor:

```
class MessageObject implements Message {
    MessageObject(Method m, Object receiver,
                  Object[] arguments);
    int timeStamp();
    Method m();
    Object receiver();
    Object[] arguments(); }
```

Creating a message could be done just like for all other objects:

```
Message msg =
new MessageObject(Method m, Object receiver,
                  Object[] arguments);
```

This implies message send in the underlying implementation. However, Message and MessageObject belong to the reflective core along with Class, Method, and Constructor. These types should be final in order to guarantee type safety at run-time. So an alternative is to have a special notation to create an asynchronous message. A functional (and hence synchronous) message is denoted using the usual dot notation:

```
x.m(a1,a2,. . .,an).
```

A non-functional (asynchronous etc.) message would be created as follows:

```
Message msg = x<=m(a1,a2,. . .,an).
```

In general, an asynchronous message does not have a result. The basic type of a message is point-to-point, one-way, and immediately executed. This type of a message could be expressed in a traditional notation

```
receiver.m(arguments)
```

In the new paradigm, the result of an asynchronous message send is a reference to the created message object. An example is:

```
Method requestQuote =
getClass(''StockBrokerAmbient'').getMethod(
    ''requestQuote'', getClass(''String''));
Message requestQuoteMsg =
    new MessageObject(requestQuote,
                      stockbrokers, stock);
```

An alternative notation looks like this:

```
Message requestQuoteMsg =
    stockbrokers <= requestQuote(stock);
```

An update message is a message that mutates the state of the receiver object and possibly other objects as well. An update message does not have a result, hence we have:

```
interface UpdateMessage extends Message { . . . }
```

A special notation for an update message is

```
x<:=m(a1,a2,. . .,an)
```

The type of this expression is UpdateMessage.

A two-way message requires a response, which communicates the result of a message. The result is produced by invoking the method future on a two-way message [23]. This method has a precondition, which is that the future is resolved, i.e., that it contains the response to the message.

```
interface TwoWayMessage extends Message{...}
```

The implementing class would contain a constructor, which takes the reply interval as one of its parameters.

```
class TwoWayMessageObject
    implements TwoWayMessage {
    TwoWayMessageObject(Method m,
                        Object receiver, Object[] arguments,
                        int replyInterval);
    boolean futureResolved();
    boolean setFuture();
    Object future()
        requires this.futureResolved();
}
```

An example of a two way message is:

```
TwoWayMessage requestQuoteMsg =
new TwoWayMessageObject(requestQuote,
                        stockbrokers, stock, 20);
```

A suggestive notation for a two way message is:

```
TwoWayMessage requestQuoteMsg =
stockbrokers <=> requestQuote(stock, 20);
```

A one-to-many message is of the type BroadcastMessage and it is sent to multiple objects. Using a suggestive notation for a one-to-many message, we would have:

```
Message requestQuoteMsg =
    stockbrokers <<=> requestQuote(stock);
```

A transient message has a discovery time specified as a finite time interval. If a message is not discovered and

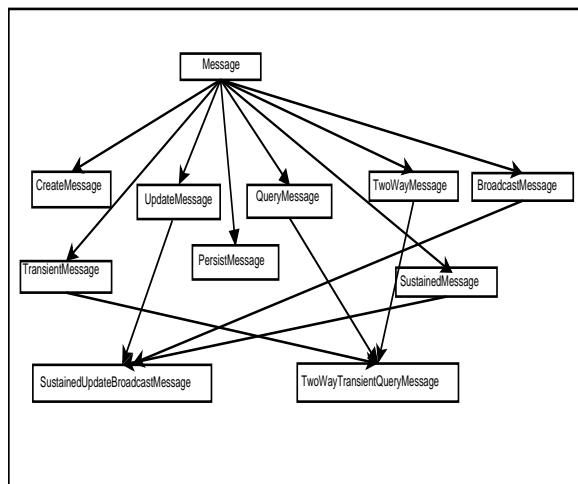


Figure 3. Message type hierarchy

scheduled for execution before its discovery time has expired, the message will be regarded as expired and will never be scheduled for execution. The discovery time will be specified in the constructor of the implementing class. A suggestive notation for a transient message is $<=|$. A sustained message (i.e., a message whose discovery time is not limited) denoted as $<=~$ is specified by a special message type *SustainedMessage*.

IV. PERSISTENT OBJECTS

An object is promoted to persistence by executing a message *persist*, which specifies a user name and a name space. This message binds the object to the given user name in the given name space. The root class *Object* is equipped with a method *persist*, which means that the model of persistence is orthogonal, i.e., objects of any type may be promoted to persistence. This is in contradistinction to the model of persistence in the mainstream object-oriented languages such as Java or C#, or the model of persistence in the ODMG (Object Data Management Group) [9], and most other object data models, which are not orthogonal.

```
class Object { . . .
void persist(NameSpace scope, String userID); }
```

A name space consists of bindings of user names to objects. Name spaces can be nested. A name space is equipped with methods for establishing such a binding and for looking up an object in a name space bound to a given user id. Typically, name spaces are persistent.

```
interface NameSpace extends ConcurrentObject{
boolean bind(Object x, String name);
Object lookup(String name);
}
```

The type *PersistMessage* is now defined as follows:

```
interface PersistMessage extends Message {
NameSpace scope();
Object userID(String name);
}
```

Creation of a persist message is denoted by a special notation using the symbols $<=!$ *persist*.

A schema extends a name space with additional methods. One of them is the method *select* that returns a set of objects in the schema that satisfy a given assertion.

```
interface Schema extends NameSpace { . . .
Set<Object> select(Assertion a);
}
```

The integrity constraints of a schema are specified in its invariant as illustrated in the example below. The schema *StockMarket* is equipped with a key constraint, a referential integrity constraint, and a value constraint.

```
interface Stock {
String code();
float price();
}
interface Broker {
String name();
Set<Stock> stocks();
}
interface StockMarket extends Schema {
Set<Stock> stocks();
Set<Broker> brokers();
invariant
keyConstraint:
(forAll s1,s2 in this.stocks():
(s1.code()==s2.code()) ==>
s1.equals(s2));
refIntegrity:
(forAll b in this.brokers():
(forAll sb in b.stocks():
(exists s in this.stocks():
(sb.code() == s.code()))));
valueConstraint:
(forall s in this.stocks():
s.price() > 0);
}
```

As for a specific assertion language, our previous results such as [3][5] are based on JML and more recent experiments are based on Spec# [2][7]. In fact, our extended virtual platform [18] accommodates a variety of assertion languages.

V. QUERIES AND TRANSACTIONS

A query message is specified below as an asynchronous message. Its type is a subtype of *TwoWayMessage*. So the result of a query may not be immediately available. When it is, it will be available by sending a functional message *future* to the query message object.

```
interface QueryMessage
extends TwoWayMessage {
Schema scope();
Assertion query();
```

```
}
```

Creation of a particular query object is illustrated below using a special notation with the symbol `<=?select:`

```
StockMarket sch; QueryMessage q;
q <=?select(
    forAll b in sch.brokers():
        (exists s in b.stocks():
            s.code()=='SNP500'));
```

A database server is a specific subtype of a service (and hence concurrent) object. It implements a schema:

```
interface DbServer
    extends ServiceObject, Schema {
    Sequence<Message> log();
}
```

Since a database server is a concurrent object, it is equipped with its own virtual machine. Typically, a database server is a persistent concurrent object. Hence by reachability, its schema (which includes persistent objects and integrity constraints) and its virtual machine will also be persistent.

A database server is equipped with a log of received messages. Here the view of messages as typed objects is critical. Committing a transaction requires extraction of the update and persist messages to reflect those changes in database collections. Implementing serializability protocols requires distinguishing update and query messages and controlling the order of their execution. All of this is possible because these messages are objects belonging to different types so that their properties can be inspected by sending functional messages to those objects.

Unlike the ODMG model [4][9], a transaction type is parametric. Its bound type specifies that the actual type parameter must be derived from the interface `Schema`.

```
interface Transaction<T extends Schema> {
    boolean commit();
    boolean abort();
}
```

Another distinctive feature of the notion of a transaction with respect to ODMG and other persistent object models is that a transaction is naturally equipped with a precondition and a postcondition and it is defined as a sequence of messages of different types (such as query, update and persist messages). The implementing class of the interface `Transaction` would have the following form:

```
class TransactionObject<T>
    implements Transaction<T extends Schema>{
    TransactionObject(T dbSchema);
    Sequence<Message> body();
    boolean commit();
    boolean abort();
}
```

Taking this approach one step further, a transaction is a concurrent object defined as follows:

```
class ConcurrentTransactionObject<T>
    implements ConcurrentObject,
    Transaction<T extends Schema>
{ . . . }
```

A few illustrative examples of transaction specification in an object-oriented assertion language are given below. A transaction `insertStock` is bound to a schema of the type `StockMarket`. The actual update has a frame specification, which states that this transaction modifies the set of stocks leaving the set of brokers unaffected. The precondition specifies that the code of the stock to be inserted is different from the codes of all existing stocks in the set of stocks. This guarantees that the key constraint will not be violated by this insertion. In addition, the precondition requires that the stock to be inserted satisfies the schema's value constraint. The postcondition guarantees that the insertion has been performed. More precisely, a stock with the code of the newly inserted stock does indeed exist in the set of stocks.

```
interface insertStock
    extends Transaction<StockMarket> {

    StockMarket schema();

    void update(Stock newStock)
        modifies stocks;
    requires
        (forall s in this.schema().stocks():
            s.code() <> newStock.code());
    requires (newStock.price() > 0);
    ensures
        (exists s in this.schema().stocks():
            s.code()==newStock.code());
}
```

The `updateStock` transaction given below performs an increase of the value of a stock with the given stock code by a given percentage. The frame constraint specifies that the transaction modifies only the set of stocks. The precondition requires that a stock with a given code does indeed exist in the set of stocks and that the percentage of increase is greater than 1. The postcondition guarantees that the stocks with the given code (there will be only one because of the key constraint) has been correctly updated.

```
interface updateStock extends
    Transaction<StockMarket> {

    StockMarket schema();

    void update(String stockCode,
                float increase)
        modifies stocks;
    requires
        (exists s in this.schema().stocks():
            s.code()==stockCode);
    requires (increase > 1);
    ensures
        (forall s in this.schema().stocks():


```

```

        (s.code() == stockCode) ==>
        (s.price() == s.price() * increase));
    }
}

```

A transaction `deleteStock` involves maintaining the referential integrity constraint, hence its frame condition specifies that the transaction modifies both the set of stocks and the set of brokers. The precondition requires that a stock with a given code does indeed exist in the set of stocks. There are two postconditions. The first one guarantees that the stock has been deleted from the set of stocks. The second postcondition guarantees that the deleted stock does not exist in the set of stocks of any broker.

```

interface deleteStock extends
    Transactions<StockMarket> {
    StockMarket schema();

    void update(Stock delStock)
        modifies stocks, brokers;
    requires
        (exists s in this.schema().stocks():
            s.code() == delStock.code());
    ensures
        (forall s in this.schema.stocks():
            s.code() <> delStock.code());
    ensures
        (forall b in this.schema().brokers():
            (forall s in b.stocks():
                s.code() <> delStock.code())));
}

```

VI. REFLECTION

Just like in Java Core Reflection (JCR), reflection in a language that supports messages as typed objects includes classes `Class`, `Method`, and `Constructor`. The main differences in comparison with JCR are:

- Reflection includes the interface `Message` with its various subtypes.
- Reflection includes the interfaces `Assertion` and `Expression` with their various subtypes.

The core reflective class `Class` has the following abbreviated signature. A distinctive feature is an assertion representing a class invariant.

```

class Class { . . .
    String name();
    Method[] methods();
    Method getMethod(String name,
                     Class[] arguments);
    Assertion invariant();
}

```

The reflective class `Method` is defined as follows. Its distinctive features are a pre condition and a post condition expressed as assertions. Their type is `Assertion`.

```

Class Method { . . .
    String name();
    Class declaringClass();
    Assertion preCondition();
}

```

```

Assertion postCondition();
Class[] arguments();
Class result();
Expression body();
Object eval(Object receiver, Object[] args);
}

```

The body of a method is an expression evaluated by the function `eval`. Just like `Assertion`, the type `Expression` belongs to the reflective core. The method `eval` evaluates the method body after binding of variables occurring in the expression representing the method body is performed. The variables to be supplied to `eval` are the receiver and the arguments.

Availability of assertions in the classes `Method` and `Class` is a major distinction with respect to the current virtual machines such as JVM or CLR (Common Language Runtime). This is at the same time a major difference with respect to the assertion languages such as JML or Spec#. Full implementation of this distinction is given in our previous work [18].

VII. VERIFICATION TECHNOLOGY

In order to carry out interactive static verification using PVS, the source object-oriented constraints must be translated into the PVS notation. PVS specifications are theories. A class equipped with constraints will be represented as a theory. Such a theory will encapsulate the underlying type along with the associated functions and predicates, and constraints will be represented as formulas in the appropriate logic. Since PVS is a higher-order system, it allows specification of specialized logics, such as temporal or separation logic. The PVS theory of schemas is equipped with a predicate `consistent`, which specifies the database integrity constraints to be redefined in a specific schema.

The transaction theory given below makes use of bounded parametric polymorphism available in PVS where the bound for the type parameter is the theory `Schema`. A transaction predicate is binary where the two arguments are the database state before and after transaction execution. The transaction predicate is a conjunction of two predicates `update` and `frame`. The `update` predicate specifies the actual effect of the transaction in transforming the database state. The `frame` predicate specifies the frame of the transaction, i.e., those components of the database state that are not affected by the transaction execution. This predicate is critical in making the task of the verifier tractable. The integrity theorem states that if the database state is consistent and a transaction is executed, the state of the database after transaction execution will be consistent.

```

Transaction[(IMPORTING Schema)
             T: TYPE FROM Schema]: THEORY
BEGIN
    Transaction: TYPE FROM Object
    schema: [Transaction -> T]

```

```

update: [T, T ->bool]
frame: [T, T ->bool]

S1,S2: VAR T
transaction(S1,S2):bool = frame(S1,S2) AND
                           update(S1,S2)

Integrity: THEOREM consistent(S1) AND
              transaction(S1,S2)
              IMPLIES consistent(S2)
END Transaction

```

A specific PVS theory for the stock market schema as specified previously in the object-oriented constraint language is given below. This theory imports theories that it needs and we do not present them in this paper. It also defines StockMarket as a type derived from the type Schema representing generic properties of all schemas. The latter contains the predicate consistent, which is redefined in the theory StockMarket as a conjunction of the key constraint, referential integrity and the value constraint. These three constraints are defined in the PVS language as specified in the theory StockMarket. stocks and brokers are functions, which return a set of stocks and a set of brokers respectively associated with a given schema.

```

StockMarket: THEORY
BEGIN
  IMPORTING Sets, Stock, Broker, Schema
  StockMarket: TYPE FROM Schema

  stocks: [StockMarket -> set[Stock]]
  brokers: [StockMarket -> set[Broker]]
  S: VAR StockMarket

  KeyConstraint(S):bool =
    (FORALL (s1,s2: Stock):
      (member(s1,stocks(S)) AND
       member(s2,stocks(S)) AND
       code(s1) = code(s2))
       IMPLIES s1=s2)

  RefIntegrity(S): bool =
    (FORALL (b: Broker):
      (member(b,brokers(S)) AND
       (FORALL (sB: Stock):
         (member(sB,bstocks(b))) IMPLIES
          member(sB,stocks(S)))))

  valueConstraint(s:Stock): bool =
    (value(s)> 0)
  valueIntegrity(S): bool =
    (FORALL (s:Stock):
      member(s,stocks(S)) IMPLIES
        valueConstraint(s))

  consistent(S):bool = KeyConstraint(S) AND
                        RefIntegrity(S) AND
                        valueIntegrity(S)
END StockMarket

```

A transaction theory InsertStock is a PVS representation of the corresponding transaction in the object-

oriented assertion language. This transaction theory imports its schema theory, and it is defined as a transaction type bound to the schema StockMarket. As in the object-oriented version, the update predicate specifies that the code of the stock to be inserted is different from all the existing codes in the set of stocks and that the new stock satisfies the schema integrity constraint. In addition, the update specifies that the set of stocks grows in size and that the new stock indeed exists in the set of stocks after the insertion transaction. The frame constraint specifies that the set of brokers is unaffected by this transaction. In addition, the frame constraint specifies that all the stocks in the initial set of stocks are still there after the transaction.

```

InsertStock: THEORY
BEGIN
  IMPORTING StockMarket,
          Transaction[StockMarket]
  InsertStock: TYPE FROM
               Transaction[StockMarket]
  S1,S2: VAR StockMarket
  s: VAR Stock
  newStock: VAR Stock

  update(newStock)(S1,S2): bool =
    (size(stocks(S2))= size(stocks(S1))+1)
    AND (FORALL s:
      (member(s,stocks(S1)) IMPLIES
       (code(s) /= code(newStock))) AND
        valueConstraint(newStock) AND
          (member(newStock,stocks(S2)))))

  frame(S1,S2): bool =
    (brokers(S1) = brokers(S2)) AND
    (FORALL s: (member(s,stocks(S1))
      IMPLIES member(s,stocks(S2))))
END InsertStock

```

The transaction theory UpdateStock given below represents the corresponding object-oriented transaction in the PVS notation. StockUpdate is defined as a type derived from the type Transaction[StockMarket], i.e., it specifies transactions associated with the schema StockMarket. The update predicate specifies that a stock with a given code exists in the set of stocks and that it has been correctly updated in the resulting set of stocks after transaction execution. The frame constraint specifies that this transaction does not affect the set of brokers nor the size of the set of stocks. In addition, it specifies that the stocks that existed initially in the set of stocks will still be there after the update transaction.

```

UpdateStock: THEORY
BEGIN
  IMPORTING StockMarket,
          Transaction[StockMarket]
  UpdateStock:
    TYPE FROM Transaction[StockMarket]
  S1,S2: VAR StockMarket
  s: VAR Stock
  increase: VAR real

```

```

update(s) (increase) (S1,S2): bool =
  (FORALL (s1,s2: Stock):
    (member(s1,stocks(S1)) AND
     code(s1) = code(s) AND
     member(s2,stocks(S2)) AND
     code(s2) = code(s)) IMPLIES
      (value(s2) = (value(s1)*increase)))
  frame(S1,S2): bool =
    (size(stocks(S2))= size(stocks(S1))) AND
     (brokers(S2) = brokers(S1)) AND
     (FORALL (s1:Stock):
       (member(s1,stocks(S1)) IMPLIES
        (EXISTS (s2:Stock):
          member(s2,stocks(S2)) AND
          (code(s1)=code(s2))))))
END UpdateStock

```

A transaction theory DeleteStock follows the above pattern except that the update and the frame predicate reflect the requirement that the referential integrity constraint of the schema StockMarket cannot be violated. The update predicate specifies that the deletion reduces the size of the set of stocks. More importantly, it specifies that the stock to be deleted actually exists in the initial set of stocks and it does not in the resulting set of stocks after deletion. Moreover, this update predicate specifies that the deleted stock does not exist in any set of stocks associated with any broker after the deletion is performed. The frame constraint specifies that the stocks with code different from the code of the deleted stock still exist in the set of stocks after deletion.

```

DeleteStock: THEORY
BEGIN
  IMPORTING StockMarket,
           Transaction[StockMarket]
  DeleteStock:
    TYPE FROM Transaction[StockMarket]

  S1,S2: VAR StockMarket
  s,s1,s2,sb: VAR Stock
  b:VAR Broker
  delStock: VAR Stock

  update(delStock) (S1,S2): bool =
    (size(stocks(S2))=(size(stocks(S1))-1))
    AND (EXISTS s: (member(s,stocks(S1)) AND
                     (code(s)=code(delStock)))) AND
    (FORALL s: (member(s,stocks(S2))) IMPLIES
     (code(s) /=code(delStock))) AND
    (FORALL b: (FORALL sb:
                 member(sb,bstocks(b)) IMPLIES
                 (code(sb) /=code(delStock)))))

  frame(delStock) (S1,S2): bool =
    (FORALL s1: (member(s1,stocks(S1)) AND
                  code(s1) /= code(delStock)) IMPLIES
     (EXISTS s2: (member(s2,stocks(S2)) AND
                  (s1=s2))))
END DeleteStock

```

VIII. RELATED RESEARCH

The orthogonal model of persistence implemented in [6] and the ODMG model of persistence [9] are based on promoting an object to persistence by either binding it to a name in a persistent name space or making it a component of an object that is already persistent. Message-based model of persistence presented in this paper is a further significantly different development after these initial approaches.

In the ODMG model, queries and transactions are objects, and so are in our model, with additional subtleties. In our approach messages are objects, and queries and updates are particular types of messages. A transaction is a concurrent object, which consists of a sequence of messages. The fact that messages are objects makes it possible to construct a transaction log as a sequence of messages of different types (queries and updates, checkpoints, commits, etc.).

General integrity constraints are missing from most persistent and database object models with rare exceptions such as [2][5][8]. This specifically applies to the ODMG model, PJama, Java Data Objects, and just as well to the current generation of systems such as Db4 Objects [11], Objectivity [15] or LINQ (Language Integrated Query) [14]. Of course, a major reason is that mainstream object-oriented languages are not equipped with constraints. Those capabilities are only under development for Java and C# [7][12].

Constraints in the form of object-oriented assertions are a key component of our approach. Database integrity constraints are specified as class invariants, transactions are specified via pre and post conditions, and queries come with general filtering (qualification) predicates. In comparison with object-oriented assertion languages, such as JML [12] and Spec# [7][13], a major difference is that in our approach assertions are integrated in the run-time type system and visible by reflection. This makes database integrity constraints accessible and enforceable at run-time. Reflective constraint management, static and dynamic techniques for enforcing constraints, and transaction verification technology are presented in [3][5][18].

Our sources of motivation for the view of concurrent, distributed and mobile objects were the languages ABCL [22][23] and AmbientTalk [10]. The core difference is that both of the above languages are untyped, whereas our approach here is based on a type system. A further distinction is that ABCL and AmbientTalk are object-based and our approach is class based. Other related work is given in [19]. Unlike ABCL reflective capabilities, reflection in this paper is type-safe. A major distinction is the assertion language as a core feature of the approach presented in this paper.

A major difference in comparison with our previous paper [1] is in the verification technology based on a higher-order verification system PVS as it applies to transaction verification.

A classical result on the application of theorem prover technology based on computational logic to the verification

of transaction safety is [20]. Other results include [8] and the usage of Isabelle/HOL [21]. Our previous results include techniques based on JML and PVS [3]. Our most recent results are based on Spec# [2]. Verification techniques of object-oriented transactions with schemas and transactions specified in either JML or Spec# are presented in [2][3].

IX. CONCLUSION

Object-oriented assertions allow specification of object-oriented schemas equipped with database integrity constraints, transactions and their consistency requirements, and queries. The view of messages as typed objects leads to a typed reflective paradigm equipped with a message-based orthogonal persistence. Reflection in this paradigm is much more general than reflection in main-stream typed object-oriented languages as it includes message and assertion types that are integrated into the run-time type system.

The presented approach requires more sophisticated users that can handle object-oriented assertion languages such as JML or Spec#. Those languages and their underlying technologies come with nontrivial subtleties as they are still in the prototype phase. Integrating these technologies into existing object database systems presents a significant challenge yet to be addressed in our future research.

On the other hand, the benefits of the availability of general constraints and static verification of transactions with respect to those constraints are very significant. Data integrity as specified by the constraints could be guaranteed. Runtime efficiency and reliability of transactions are significantly improved. Expensive recovery procedures will not be required for constraints that were statically verified. In addition, more general application constraints that are not necessarily database constraints could be guaranteed. All of this produces a much more sophisticated technology in comparison with the existing ones.

REFERENCES

- [1] S. Alagić and A. Yonezawa, Ambients of persistent concurrent objects, Proceedings of DBKDA 2011 (Advances in Databases, Knowledge, and Data Applications), pp. 155-161, IARIA 2011.
- [2] S. Alagić, P. Bernstein, and R. Jairath, Object-oriented constraints for XML Schema, Proceedings of ICOODB 2010, *Lecture Notes in Computer Science 6348*, pp. 101-118.
- [3] S. Alagić, M. Royer, and D. Briggs, Verification technology for object-oriented/XML transactions, Proceedings of ICOODB 2009, *Lecture Notes in Computer Science 5936*, pp. 23-40.
- [4] S. Alagić, The ODMG object model: does it make sense?, Proceedings of OOPSLA, pp. 253-270, ACM, 1997.
- [5] S. Alagić and J. Logan, Consistency of Java transactions, Proceedings of DBPL 2003, *Lecture Notes in Computer Science 2921*, pp. 71-89, Springer, 2004.
- [6] M. Atkinson, L. Daynes, M. J. Jordan, T. Printezis, and S. Spence, An orthogonally persistent JavaTM, ACM SIGMOD Record 25, pp. 68-75, ACM, 1996.
- [7] M. Barnett, K. R. M. Leino, and W. Schulte, The Spec# programming system: an overview, Microsoft Research 2004, <http://research.microsoft.com/en-us/projects/specsharp/> [retrieved: December 31, 2011].
- [8] V. Benzaken and X. Schaefer, Static integrity constraint management in object-oriented database programming languages via predicate transformers, Proceedings of ECOOP '97, *Lecture Notes in Computer Science 1241*, pp. 60-84, 1997.
- [9] R. G. G. Cattell, D. Barry, M. Berler, J. Eastman, D. Jordan, C. Russell, O. Schadow, T. Stanienda, and F. Velez, *The Object Data Standard: ODMG 3.0*, Morgan Kaufmann, 2000.
- [10] T. Van Cutsem, Ambient references: object designation in mobile ad hoc networks, Ph.D. dissertation, Vrije University Brussels, 2008.
- [11] Db4 objects, <http://www.db4o.com> [retrieved: December 31, 2011].
- [12] G. T. Leavens, E. Poll, C. Clifton, Y. Cheon, C. Ruby, D. Cook, P. Muller, and J. Kiniry, JML Reference Manual, <http://www.eecs.ucf.edu/~leavens/JML/> [retrieved: December 31, 2011].
- [13] K. R. Leino and P. Muller, Using Spec# language, methodology, and tools to write bug-free programs, Microsoft Research, <http://research.microsoft.com/en-us/projects/specsharp/> [retrieved: December 31, 2011].
- [14] LINQ: Language Integrated Query, <http://msdn.microsoft.com/en-us/library/bb308959.aspx> [retrieved: December 31, 2011].
- [15] Objectivity, <http://www.objectivity.com/> [retrieved: December 31, 2011].
- [16] S. Owre, N. Shankar, J. M. Rushby, J. Crow, and M. Srivas, A tutorial introduction to PVS, <http://www.csl.sri.com/papers/wift-tutorial/> [retrieved: December 31, 2011].
- [17] S. Owre, N. Shankar, J. M. Rushby, and D. W. J. Stringer-Clavert: PVS Prover Guide, SRI International, Computer Science Laboratory, Menlo Park, California <http://pvs.csl.sri.com/doc/pvs-prover-guide.pdf> [retrieved: December 31, 2011].
- [18] M. Royer, S. Alagić, and D. Dillon, Reflective constraint management for languages on virtual platforms, *Journal of Object Technology*, vol 6, pp. 59-79, 2007.
- [19] J. Schaefer and A. Poetzsch-Heffter, JCoBox: Generalizing active objects to concurrent components, Proceedings of ECOOP 2010, *Lecture Notes in Computer Science 6183*, pp. 275-299.
- [20] T. Sheard and D. Stemple, Automatic verification of database transaction safety, *ACM Transactions on Database Systems* 14, pp. 322-368, 1989.
- [21] D. Spelt and S. Even, A theorem prover-based analysis tool for object-oriented databases, *Lecture Notes in Computer Science 1579*, pp 375 - 389, Springer, 1999.
- [22] T. Watanabe and A. Yonezawa, Reflection in an object-oriented concurrent language, Proceedings of OOPSLA, pp. 306-315, ACM Press 1988.
- [23] A. Yonezawa, J.-P. Briot, and E. Shibayama, Object-oriented concurrent programming in ABCL/1, Proceedings of OOPSLA, pp. 258-268, ACM Press 1986.

Models of 40-Year Spatial Development of Cities in the Czech Republic in a geographic information system

Lena Halounová, Karel Vepřek, Martin Řehák
 Dept. of Mapping and Cartography
 Faculty of Civil Engineering, CTU in Prague
 Prague, Czech Republic

e-mail: lena.halounova@fsv.cvut.cz
arch.veprek@tiscali.cz
martin.rehak.1@fsv.cvut.cz

Abstract—There are many indicators of sustainable development of towns defined by urban specialists, sociologists, economists, etc. The paper presents the first part of a project whose goal is to find indicators of harmonic development of towns based on analysis of forty years development of fifty Czech towns. The indicators are studied in land use spatial changes, demography and road traffic intensity changes. First ten towns were processed for the period between 1970 and 2009 being mapped in general urban land use classes and related to the measured road density. City land use class areas were derived from combination of actual and historical city plans and remote sensing data using geographic information system tools. It was found that the traffic intensity within towns and to and from towns is more dependent on existence of close highways and by-pass roads unlike number of inhabitants, e.g. Political changes from the communist regime to the democratic one was also an important breakpoint in the city developments. Increase of the road traffic intensity and enlarging of residential areas are features proving the fact. The paper presents a methodology of spatial mapping of land use classes utilized for determination of town development. The city developments and their relation to road traffic are documented on maps and graphs.

Keywords-GIS; remote sensing data; city plan; number of inhabitants; urban model; land use; road traffic intensity

I. INTRODUCTION

The development of cities during last decades has faced us with a new situation. Most inhabitants in many European countries are concentrated in large towns. One fifth of the Czech Republic population is living in three largest towns – Prague, Brno and Ostrava. Fig. 1 shows percentage of inhabitants of all analyzed cities compared to two largest cities of the country- Prague and Brno - forming nearly half of inhabitants; therefore, the results of the analysis can be regarded as really representative.

Present state of the balance among consumption level of society and quality of life is a matter of scientific papers, research [2][5][6], many projects [1][7], and political and economical discussions in many countries. Life quality is directly related to a lot of environmental and socio-economic conditions. These conditions determine a harmonic development which should be based on equivalent and adequate demands of the human society. To define “adequate” means to take into account both consumption,

and quality of life. Both are closely connected to the road traffic and its intensity.

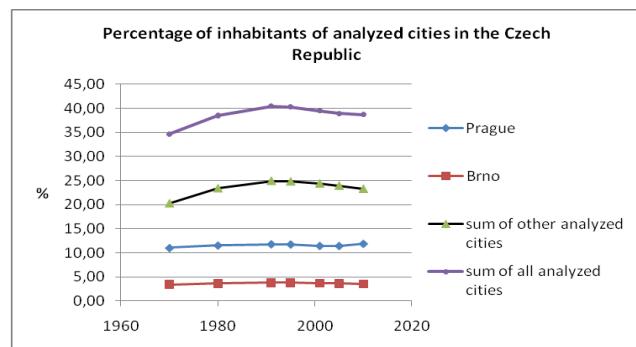


Figure 1. Development of percentage of inhabitants of analyzed cities in the framework of the Czech Republic

The Department of Mapping and Cartography has been processing a COST project focused on a detailed evaluation of relations between the quality of life and present behavior of the human society. The project goal is to create a model allowing improving the present development status in urban areas being less demanding and ensuring their sustainable development.

The project is a logical continuation of several projects performed by specialists from the Czech Technical University (CTU) in Prague in the Czech Republic and the State Institute for Regional Planning who have collected and summarized large data volume of fifty towns (including three largest ones - Prague, Brno and Ostrava) on:

1. Functional typology comprising five general land use classes: housing and infrastructure areas summarized into a residential one, industrial and agricultural production areas named areas of production, areas of traffic, areas of recreation including sport and green vegetation land, surface water areas, etc., and areas of other functions like arable land, orchards, meadows, technical infrastructure – waste water treatment plant areas, quarries, e.g., this classification is used by urban planners and the individual city land use maps utilize its list.
2. The second group of land use has been collected and recorded by the Czech Office for Surveying,

Mapping and Cadastre (COSMC) and Czech Statistical Office and consists of urban, agriculture, forest and water surface areas and other function areas. The data are related to 1970, 1980, 1990, 2000, 2005, and 2008.

3. Basic components of the environment.
4. Basic components of the social and economical development.

The last two groups of data were collected within many diploma theses for individual cities as the statistical data were recorded for districts only.

The previous projects were focused on statistical data collected from the above mentioned sources and their processing. They did not comprise any spatial data and no spatial analyses were performed. Their city collection resulted in a large range of cities differing by population (from seventeen thousand to more than one million inhabitants), by economic orientation (agricultural, industrial, university, touristic), by natural conditions (lowland surrounded by agricultural areas, mountainous situated large forest areas), by geographic position in the republic – close/far to a frontier, etc. The city set is a good sample covering practically all Czech city types.

The processed project is focused on two new views – to select suitable indicators of the sustainable city development in the Czech Republic using also spatial characteristics together with already collected ones, and the role of the road traffic intensity in the development of cities and their mutual relation and impacts.

Individual land use areas offer different conditions for living. The same land use classes in different areas and therefore all spatial units are characterized by a long list of attributes.

II. CZECH CITIES AND THEIR DEVELOPMENT

The Czech Republic does not have a continuous political and urban development. The development was formed mainly by political decisions having a decisive role of the urban land use changes. After the Second World War the urban development was relatively uneven and can be characterized by three types of cities. One type of cities had only a relatively slow and continuous spatial evolution within their administrative boundaries. The second type are cities with growing administrative areas; however, this growth was artificial as a result of political decisions to join surrounding villages to a close city. This joining was in two phases in 70-ies and 1989. The third group of towns is similar to the second one; the only difference is in their further separation of one or more early joined villages. The separations occurred after changing of the political regime in the country at the end of 1989 from the communist to the democratic regime. This development definition was firstly described and denoted by Vepřek [8]. He uses three new terms: core area for town size representing in most cases a status in 70-ies of the 20th century (1974, 1976). These were years when the process of joining villages to neighbor cities

became an important phenomenon in administrative structure of the country. The joined areas are named associated areas by Vepřek in [8]. Urban parts in associated regions are denoted agglomerated ones by Vepřek in [8]. The parts, which became independent villages, are called peripheral areas by Vepřek in [8].

These spatial developments are archived in Cadastre books in the form of table records showing concerned cadastre districts and also in COSMC data available on its portal as excel files. The transfer of this information into spatial data can show the spatial city development using the cadastre districts' boundaries of an appropriate period. This transfer was performed also into city plans, whose processing intervals vary in individual towns. City size evolutions cannot be derived from remote sensing data. If a town belongs to the second or third group, there are large spatial changes. The largest parts of these changes are in prevailing part represented by agricultural areas. The main difference between a core area and associated area are separated urban parts occurring in them.

III. LAND USE MAPS

The forty years spatial development is characterized (in the project) by land use maps of functional classes in 2009, 2000, 1990, 1980 and 1970. High number of cities covering large areas demanded to use time effective method for their mapping. Their creation was based on adaptation of city plans, remote sensing and statistical data.

Each urban area has a city plan as given by the Czech Law. Land use class determinations were done from land use classes applied by urban engineers in city plans which have been processed in several-year periods differently in individual cities. Therefore the city plans were the first information layer for urban land use maps processing. The plans comprise maps of the current land use/cover state and proposals for the future development and it was necessary to separate the urban plans. The process started by creation of the latest year map and ended by the 1970 land use map.

A. Determination of Land Use Classes

The first step of creating of the first time level of each land use map was a reclassification of the original city plan classes. The city map legends are not standardized in the country, however, their class lists are more detailed than the classes used in the project. The reclassification reflects both five above mentioned land use classes and the COSMC land use classes distinguishing different ones: built-up, agricultural, forest, water, and other areas.

The final functional classes were residential, production, recreational, traffic and other areas. Each class is therefore formed by a higher number of city plan classes. The residential area is formed by mixed residential region, general residential and rural ones, and public areas, e.g. The reclassification means also including local roads belonging to roads of low level in the state road hierarchy into residential or other surrounding areas. The reclassification is

performed individually for each town according to its city plan classes.

The advantage of this approach was the fact that the basic classification was performed by urban specialists.

B. Land Use Mapping

As the city plans comprise not only a real status, but also an urban plan (as it was mentioned above), the next step was to verify the present city plans and the real state of cities as they can and really significantly differ from the real state especially in newly urbanized areas. The second reason for the verification is also the city plans' date of origin. This part of the processing was done by visual interpretation of the remote sensing data (aerial photographs) combined with a change registration/vectorization of the vector city plan and the result was a map of functional classes of the present state. The first map = the latest one (2009) was a result of the present city plan processing by implementing corrections found in discrepancies between the plan and aerial orthophotographs.

The previous time level of the land use map was derived from the present land use map copy by comparison with aerial photographs collected in between 1950 and 2000, and satellite image data (Thematic Mapper, MSS data) covering time span from 1970 to 2000. The satellite data were used for detection of land use changes between two time levels [6]. They were derived from a subtraction of original satellite image bands and normalized vegetation index in two different years (the 2008 band minus 2000 band, e.g.) and thresholding of the subtracted bands making the verification easier and quicker. Found changes were pixels with high positive or negative values. This approach yielded areas with different land cover, however, there was an additional task to determine and "translate" each land cover change into appropriate land use change. Each functional class comprises a wide range of the land cover classes in the aerial photographs spatial resolution; however, these detailed classes are not in prevailing part detectable on the Thematic Mapper data. The Thematic Mapper resolution does not allow determining urban functional classes – agriculture area spectral behavior can be similar to vegetated areas for some plants, etc. The areas with important changes (extreme positive and negative pixel values) were verified using the aerial photograph taking into account also their shape and texture. The oldest map showing the 1970 year was also in certain cases visually controlled using aerial orthophotomosaic created from aerial orthophotographs collected in 50-ies in the last century.

The principle of mapping based on the latest land use map as the first processed level and further steps heading back to the previous levels using always copy of the "younger" processed period as a base map for the "older" = previous time level proved to be the most effective. The current city plans' and present land use state differences were not so numerous. There was another advantage of this approach; it allows ensuring the correct topology of each land use development map set, detected classes and their evolution.

All functional classes were controlled by the statistical table data available at the Czech Office for Surveying, Mapping and Cadastre for city administrative areas. The functional classes in individual years were used for further evaluation between the road transport density, town development and investments into road network in the form of new by-pass, highways, etc. The indicators showing the relation were the following: development of functional class areas, development of number of inhabitants, development of road transport density, and building of new decongesting roads.

IV. ROAD TRAFFIC DATA

A large data base of the road network development has been created by the Road and Motorway Directorate. The data base comprises - among others - measurements of the road traffic intensity in many points of roads of various classes since 60-ies of the 20th century (1968, 1973, 1980, 1990, 1995, 2000, 2005, and 2010). The road traffic intensity is a number of vehicles per 24 hours, which passed through a determined point on a road in both directions. It is an average of several 24 hours' data collection.

The measurements are available in map forms where each location is marked together with total amount of passed vehicles (including motorcycles), and tables where the amount is enumerated in a more detailed way distinguishing heavy-duty vehicles, cars and motorcycles.

V. SOCIO-ECONOMIC DATA

A deep analysis of another large data volume which has been collected since the second half of the 20th century will be performed in the proposed project. The data comprise 10-year research of socio-economic data of environmental changes performed at the University of Economics in Prague, e.g. Each city is described by several hundreds of statistical data. The data were collected by many students of the university within their theses. The processing of the data is not presented in this paper and is a matter of the further research.

VI. RESULTS

First twenty cities processed in the first year and a half of the project have brought very interesting results.

Kladno is one of processed towns situated 30 km north west from Prague. The town consists both of a core, and associated parts. The city was an industrial city in the communist period of the republic. The industrial production has been extremely declining since 1990 and most inhabitants are employed in Prague at present. Fig. 2 shows spatial changes of four functional classes between 1969 and 2009 mapped by the above mentioned method. The dashed line (Fig. 2a) determines the core area as an administrative city boundary at the end of 70-ies in the 20th century. The solid line delineates the administrative city boundary since 80-ies of the previous century which has not changed. The red color patches are residential areas in 1969. The green patches are residential areas built between 1969 and 2008 [3].

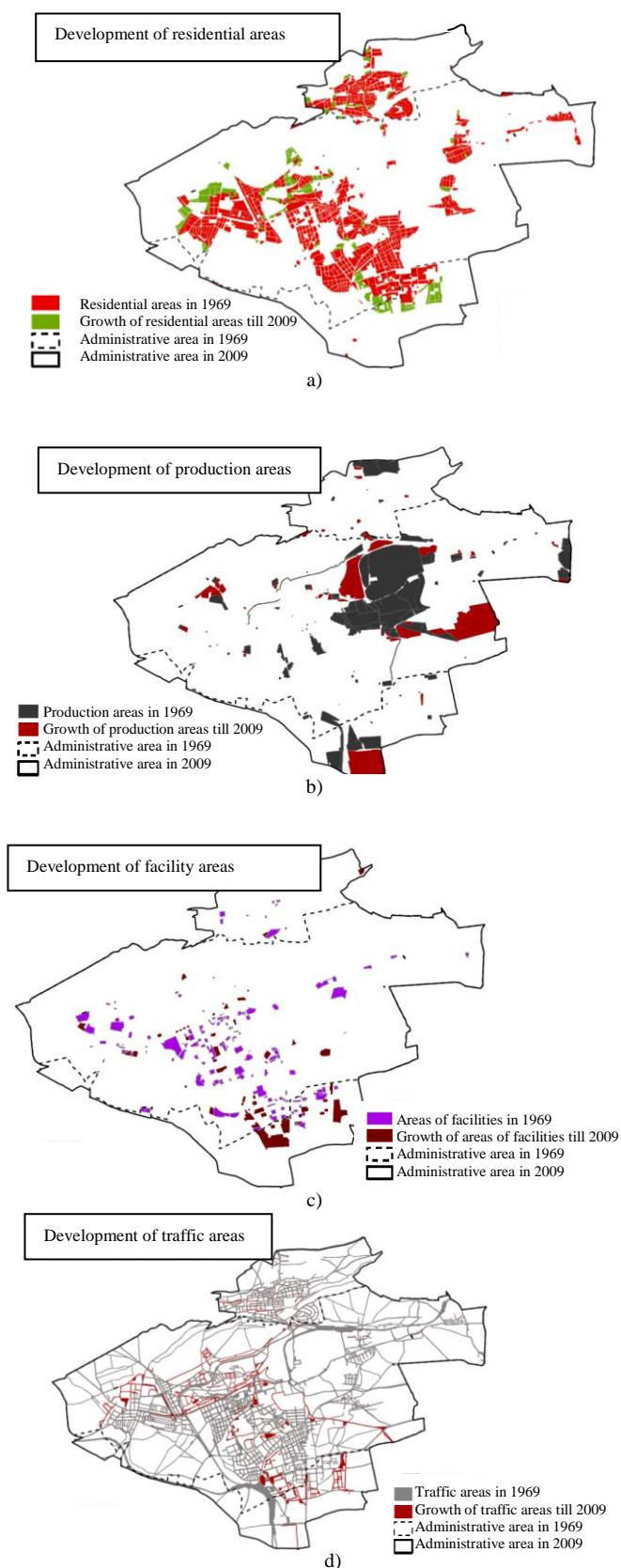


Figure 2a, b, c, d. Development of the Kladno functional classes for the 1969 – 2009 period

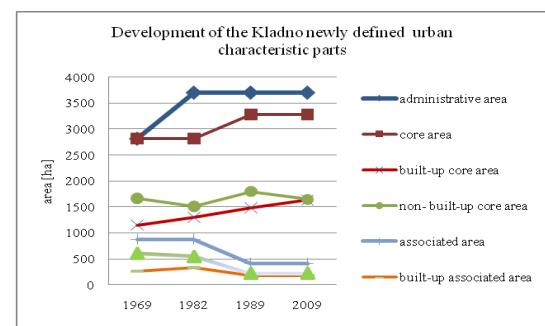


Figure 3. Example of spatial development of Kladno during last 40 years shown in administrative, core, associated, built-up and non-built-up areas

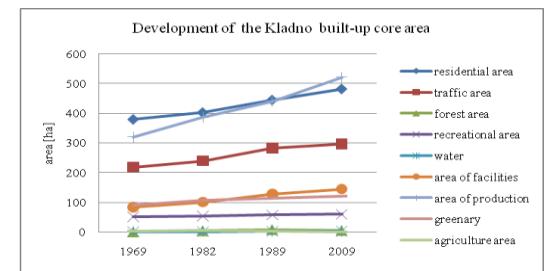


Figure 4. Example of spatial development of Kladno during last 40 years shown in land use classes

Development of production areas is shown in Fig. 2b, facility areas in Fig. 2c), and traffic areas in Fig. 2d.

Town development is further presented on two graphs – Fig. 3 and Fig. 4. Fig. 3 shows administrative, core and associated areas with their built-up and non-built-up area. Fig. 4 represents built-up core area. Looking at the statistical evaluation presented in Fig. 4, we can see that it was the area of production whose growth was the steepest in the core part. Comparing residential parts development, we can find that it covers larger areas with a steeper increase of size than those of traffic ones within the core region during last 20 years. However, there is a new highway passing the town in 5 km distance enabling the town to be used in prevailing part as a terminal location for the road traffic and not as a passing through town location in direction between Prague and north-west. The town has not yield larger areas for recreation, leisure time, sport, etc., during last 40 years (Fig. 4).

The administration area has not changed since 1982. The core area changed - unlike most towns - between 1982 and 1989. The residential area and area of production cover a similar part of the built-up area, however, the growth of the production area is steeper. Non built-up areas are in a prevailing part the forested ones. The associated areas are in most cases formed by an agricultural and forest land, however, their sizes decline after the 1989 political change. The spatial changes are described in the coincidence table

(Table I.). Each row shows the size of an individual class and its transformation between 1950 and 2008. Each column comprises original areas forming the present size of an individual class. Areas without changes are highlighted in diagonal cells of the table. Comparing of the last row (Sum 2008) and last column (Sum 1950) allows to find increase and decrease of class areas.

TABLE 1. THE COINCIDENCE TABLE SHOWS CHANGES BETWEEN 1950 AND 2008. RESIDENTIAL AREAS HAVE NOT CHANGED ON 207 HECTARES AND 25 % OF RESIDENTIAL AREAS HAS TRANSFORMED TO THE TRAFFIC, OTHER, PUBLIC, FACILITY, PRODUCTION, GREEN, AND AGRICULTURE AREAS (SEE THE RESIDENTIAL ROW). ON THE CONTRARY, THE PRESENT STATE OF THE RESIDENTIAL AREA IS NEWLY (AFTER 1950) FORMED BY TRAFFIC, OTHER, PUBLIC, GREEN, AGRICULTURE AREAS AND ENLARGED ON 130 % OF THE ORIGINAL SIZE (363 HA) (EXAMPLE OF THE CITY OF MĚLNÍK)

Land Use Classes	Land Use Classes											
	Residential	Traffic	Forest	Other	Recreation	Public	Water	Facility	Production	Green	Agriculture	Sum 1950
residential	207,47	5,56	0,00	13,06	0,00	11,79	0,00	8,32	16,94	9,98	4,01	277,14
traffic	6,24	34,55	0,95	0,34	0,16	9,40	0,00	0,21	1,83	2,86	8,16	64,97
forest	0,00	0,13	58,72	0,00	0,41	0,13	0,00	0,00	0,00	3,35	0,05	63,25
other	2,75	0,17	0,00	5,81	0,00	1,28	0,00	0,06	3,72	1,21	1,33	16,33
recreation	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
public	9,79	9,76	0,19	2,91	0,03	24,55	0,00	1,09	4,40	10,02	11,18	73,92
water	0,00	0,05	0,00	0,00	0,00	0,27	64,09	0,00	2,39	0,96	0,34	68,10
facility	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
production	0,68	1,34	0,00	3,61	0,00	2,87	1,50	0,00	66,55	3,54	0,00	80,09
green	14,09	5,47	0,00	6,94	1,63	10,40	0,00	3,27	8,66	123,28	14,66	188,88
agriculture	122,17	25,98	0,88	48,32	17,80	57,45	0,01	16,04	102,99	176,20	1082,75	1659,68
sum 2008	363,19	83,01	60,74	80,99	20,03	118,14	65,60	28,99	207,48	331,40	1122,48	

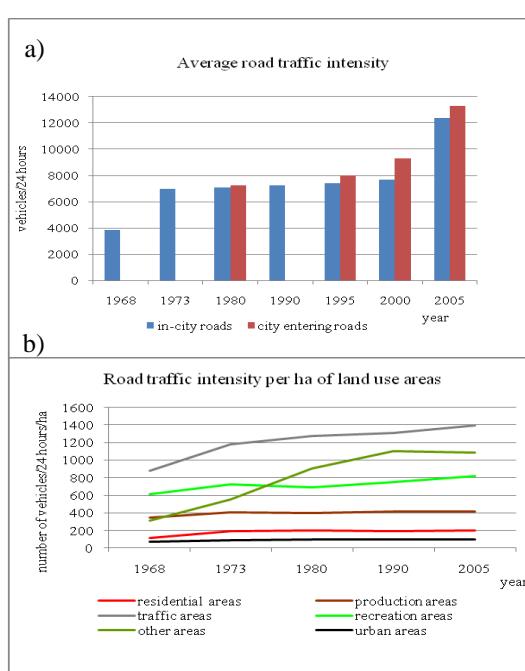


Figure 5. Sum of all measured segments on higher level roads used also for passing traffic and on local roads (a). Traffic intensity calculated as a ratio of all vehicles per 24 hours and size of functional areas (b)

The road traffic intensity was checked both on local and higher level class roads. Both road types express a growth, however, mutually uncomparable. The slope of the growth is lower after 1995 when a new pass-by highway was built (Fig. 5a). This phenomenon is presented as an impact of the highway construction out of the city on traffic intensity of the individual functional land use classes in Fig. 5b.

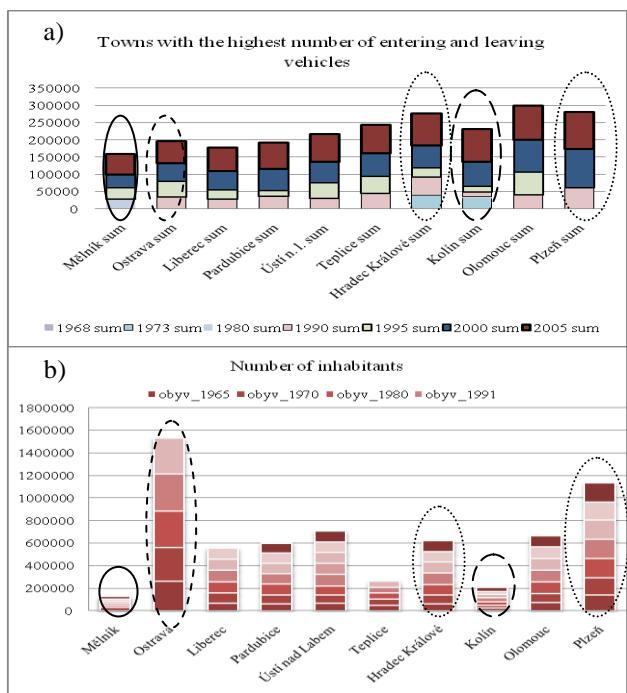


Figure 6 a, b. Comparing of the traffic intensity (a) since 1968 to 2005 and number of inhabitants (b) in similar periods (till 1991)

The traffic intensity was compared to land use classes in individual analyzed years.

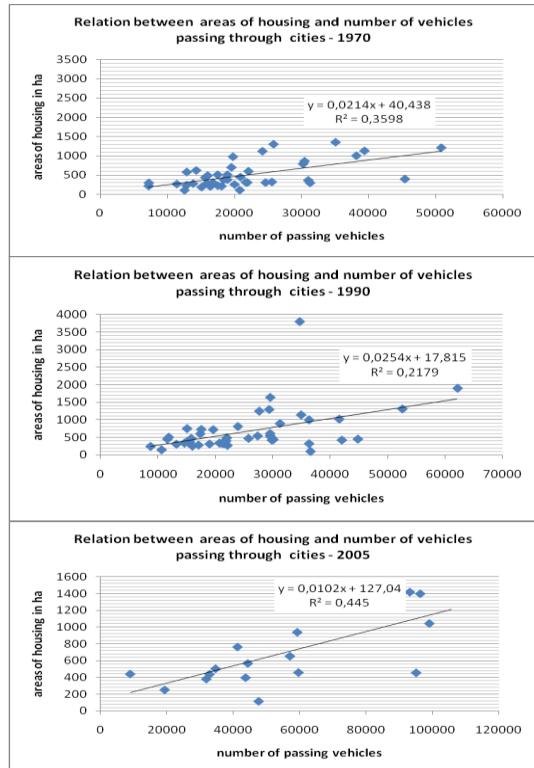


Figure 7. Graphs showing relation between number of vehicles passing through analyzed cities and their area of housing

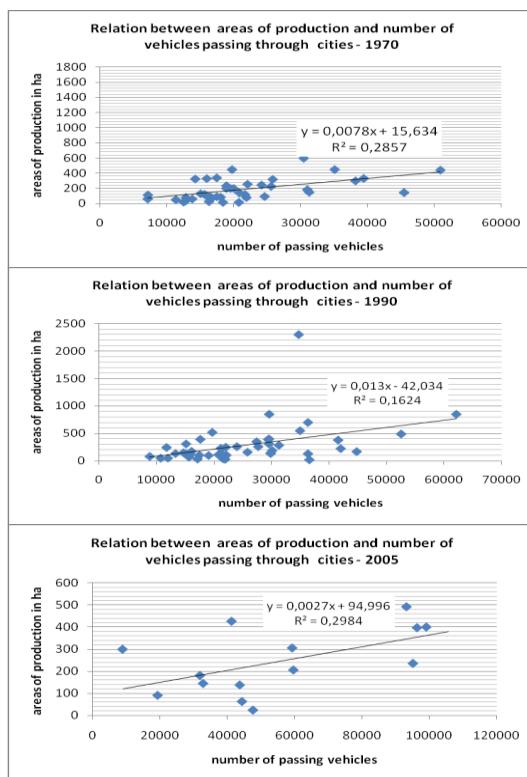


Figure 8. Graphs showing relation between number of vehicles passing through analyzed cities and their area of production

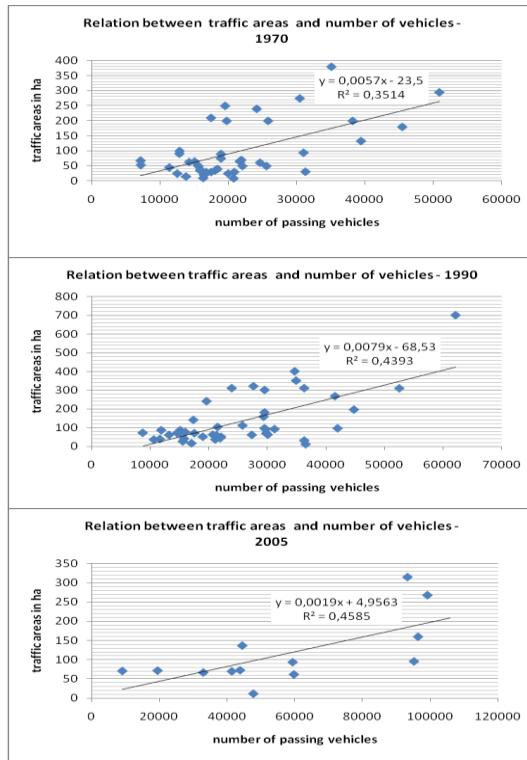


Figure 9. Graphs showing relation between number of vehicles passing through analyzed cities and their traffic area

TABLE II. CORRELATION COEFFICIENT OF RELATION BETWEEN ROAD INTENSITY OF PASSING VEHICLES AND AREAS OF VARIOUS LAND USE

Correlation coefficient between road traffic intensity and areas					
	urban areas	areas of housing	areas of production	areas of production and housing	traffic areas
1970	0,62	0,60	0,53	0,62	0,59
1980	0,34	0,35	0,24	0,33	0,57
1991	0,44	0,47	0,40	0,46	0,66
2005	0,70	0,67	0,54	0,67	0,68

The graphs on Figs. 7, 8 and 9 and Table II show that the best correlation between road traffic intensity and land use class occurs at traffic areas. These are areas of housing that have higher correlation coefficient on traffic intensity than areas of production. The best correlation for all land use areas has been found for 2005.

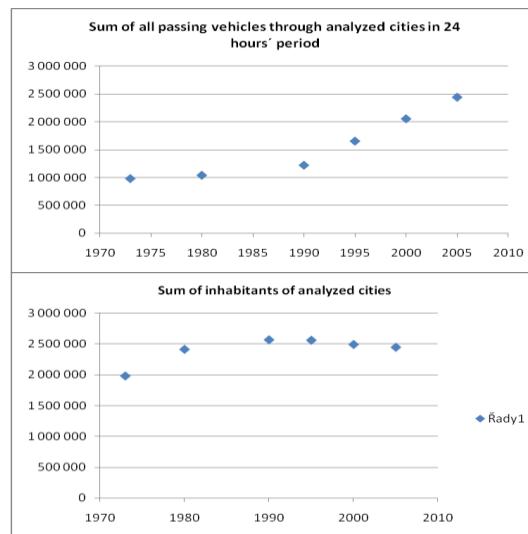


Figure 10. Development of number of passing vehicles through cities and their sum of inhabitants

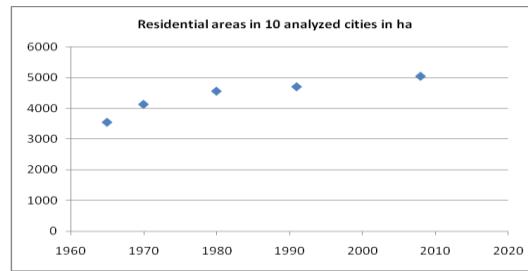


Figure 11. Development of residential areas in 10 analyzed cities

VII. CONCLUSIONS

The project methodology is based on a multi-correlational processing using statistical (social, economical, geographical and natural) data of cities and spatial land use

development and changes. There are more than six hundred economical, social and other statistical indicators whose mutual relation are prepared to be analyzed. The spatial land use change developments are visualized by the coincidence table (Table I.) and were used as the first group of indicators. Their correlation coefficients are in Table II.

Relations between town development and road traffic density showed interesting dependences. The traffic intensity changes cannot be generalized for a town as a unit. There are serious temporal changes within each town. These changes are caused by newly built commercial areas where this growth is incomparable to any other locations in the city, by new roads passing out of cities, e.g. The road traffic is also an important indicator of the economic inhabitant level – the traffic increase of personal and heavy-duty vehicles intensity on one side and decrease of motorcycle intensity and number of inhabitants on the other side since 70-ies is a proof of the higher economical power of city inhabitants which was found at all fifty analyzed cities. The fact is also documented on Fig. 10 by increasing sum of passing vehicles and decreasing number of inhabitants. The higher economical power proves also Fig. 11 accompanying Fig. 10 and showing the growth of residential areas for the decline of inhabitants.

Results of any urban planning are always long lasting phenomena influencing the society. The spatial land use changes in 50 various cities will yield a rich source of data for the evaluation. The road traffic intensity is information on the air pollution; data on life expectancy are an issue concerning a social situation and health care, etc. The project results should offer a set of usable tools = indicators for urban planners and their further urban planning to achieve sustainable development of cities in the Czech Republic.

The paper presents a starting part of analysis performed for functional classes and road traffic intensity. Spatial changes and their relation to the traffic evolution have already brought a great deal of information which will be processed in a form of one of indicators.

The first part of the research was presented during the IARIA GEOProcessing 2011 [3].

The future research is focused on determining a list and sequence of indicators for the sustainable development of cities. The available data will be processed for selection of substantial indicators and their influence. There are still spatial ones, which have not been utilized as – distance to frontiers, distance to larger cities, etc. except for mainly socio-economical indicators that will be analyzed.

ACKNOWLEDGMENT

The paper is financed by two projects: Modeling of urban areas to lower negative influences of human activities project of the Ministry of Education (OC10011) of the Czech Republic and the Management of sustainable development of life cycle of constructions, civil engineering firms and

regions of the Ministry of Education project (VZ 05 CEZ MSM 6840770006) of the Czech Republic.

REFERENCES

- [1] Global City Indicators Program, <http://www.cityindicators.org/>; cit. 07/02/2012.
- [2] Brugmann, J., "Is there a method in our measurement? The use of indicators in local sustainable development planning", Local Environment, vol. 2, issue 1, February 1997, pp. 59 – 72.
- [3] Halounová, L., Vepřek, K., Řehák, M., "Geographic Information System Models of 40-Year Spatial Development of Towns in the Czech Republic", The third International Conference on Advanced Geographic Information Systems, Applications, and Services, GEOProcessing 2011, Guadeloupe, France, February 23-28, 2011.
- [4] Howell, E., Pettit, K.L.S., Ormond, B.A., and Kingsley, G.T., "Using the National Neighborhood Indicators Project to Improve Public Health, Journal of Public Health Management and Practice", vol. 9, May/June 2003, pp. 235-242.
- [5] Kingsley, G.T., Kathryn, L.S., and Pettit, K.L.S., "Neighborhood Information Systems: We Need a Broader Effort to Build Local Capacity", Metropolitan Housing and Communities Policy Newsletter, October 2004.
- [6] Lillesand, T.M., Kiefer, R.W., and Chipman, J.W., "Remote Sensing And Image Interpretation", 5th Ed., Wiley, 2010.
- [7] United Nations Human Settlements Programme, UN-HABITAT, Kenya, http://ww2.unhabitat.org/programmes/guo/urban_indicators.asp, 2003, cit 07/02/2012.
- [8] Vepřek, K. et al., "Analysis of 100 years urban development of Hradec – Pardubice regional agglomeration focused on detection of general tendencies and regularity" - research project VÚP No16-521-503, Terplan Praha, 1983.

Rainbow Table Optimization for Password Recovery

Vrizlynn L. L. Thing, Hwei-Ming Ying

Cryptography & Security Department
 Institute for Infocomm Research, Singapore
 {vriz,hmying}@i2r.a-star.edu.sg

Abstract—As users become increasingly aware of the need to adopt strong password, it also brings challenges to digital forensics investigators due to the password protection of potential evidence data. In this paper, we discuss existing password recovery methods and present a design of a time-memory trade-off pre-computed table coupled with a new sorting algorithm. We also propose 2 new storage methods and evaluated their performance based on storage conservation and success rate improvement. Considering both alpha-numeric passwords and passwords consisting of any printable ASCII character, we show that we are able to optimize the rainbow table performance through an improvement of up to 26.13% in terms of password recovery success rate, and an improvement of up to 28.57% in terms of storage conservation, compared to the original rainbow tables.

Keywords - Digital forensics, password recovery, rainbow table optimization, time-memory trade-off, cryptanalysis.

I. INTRODUCTION

In computer and information security, the use of passwords is essential for users to protect their data and to ensure a secured access to their systems/machines. However, in digital forensics, the use of password protection presents a challenge for investigators while conducting examinations. As mentioned in [3], compelling a suspect to surrender his password would force him to produce evidence that could be used to incriminate him, thereby violating his Fifth Amendment right against self-incrimination. Therefore, this presents a need for the authorities to have the capability to access a suspect's data without expecting his assistance. While there exist methods to decode hashes to reveal passwords used to protect potential evidence, lengthier passwords with larger characters sets have been encouraged to thwart password recovery. Awareness of the need to use stronger passwords and active adoption have rendered many existing password recovery tools inefficient or even ineffective.

The more common methods of password recovery techniques are guessing, dictionary, brute force and more recently, using rainbow tables. The guessing method is attempting to crack passwords by trying “easy-to-remember”, common passwords or passwords based on a user's personal information (or a fuzzy index of words on the user's storage media). A statistical analysis of 28,000 passwords recently stolen from a popular U.S. website revealed that 16% of the users took a first name as a password and 14% relied on “easy-to-remember” keyboard combinations [4]. Therefore, the guessing method

can be quite effective in some cases where users are willing to compromise security for the sake of convenience.

The dictionary attack method composes of loading a file of dictionary words into a password cracking tool to search for a match of their hash values with the stored one. Examples of password cracking tools include Cain and Abel [5], John the Ripper [6] and LCP [7].

In the brute force cryptanalysis attack, every possible combination of the password characters is attempted to perform a match comparison. It is an extremely time consuming process but the password will be recovered eventually if a long enough time is given. Cain and Abel, John the Ripper as well as LCP are able to conduct brute force attacks.

In [8-11], the authors studied on the recovery of passwords or encryption keys based on the collision of hashes in specific hashing algorithms. These methods are mainly used to research on the weakness of hashing algorithms. They are too high in complexity and time consuming to be used for performing password recovery during forensics investigations. The methods are also applicable to specific hashing algorithms only.

In [12], Hellman introduced a method, which involves a trade-off between the computation time and storage space needed to recover the plaintext from its hash value. It can be applied to retrieve Windows login passwords encrypted into LM or NTLM hashes [13], as well as passwords in applications using these hashing algorithms. Passwords encrypted with hashing algorithms such as MD5 [14], SHA-2 [15] and RIPEMD-160 [16] are also susceptible to this recovery method. In addition, this method is applicable to many searching tasks including the knapsack and discrete logarithm problems.

In [17], Oechslin proposed a faster cryptanalytical time-memory trade-off method, which is an improvement over Hellman's method. Since then, this method has been widely used and implemented in many popular password recovery tools. The pre-computed tables that are generated in this method are known as the rainbow tables.

In [18], Narayanan and Shmatikov proposed using standard Markov modeling techniques from natural language processing to reduce the password space to be searched, combined with the application of the time-memory trade-off method to analyse the vulnerability of human-memorable passwords. It was shown that 67.6% of the passwords can be successfully recovered using a 2×10^9 search space. However, the limitation of this method is that the passwords were assumed to be

human-memorable character-sequence passwords.

In this paper, we present a new design of an enhanced rainbow table [1,2] by proposing a novel time-memory trade-off pre-computed table structure and a rainbow table sorting algorithm. Maintaining the core functionality of the rainbow tables, we optimized the storage space requirement while achieving the same success rate and search speed.

The rest of the paper is organized as follow. In Section 2, we present a discussion on the existing time-memory trade-off password recovery methods. We then give an overview of our enhanced rainbow table design in Section 3. We describe the design of our sorting algorithm in Section 4. Analysis and evaluation are presented in Section 5. The description of the 2 new proposed storage methods is provided in Section 5 due to the importance of their considerations during evaluations. Conclusions follow in Section 6.

II. ANALYSIS OF EXISTING WORK

The idea of a general time-memory tradeoff was first proposed by Hellman in 1980 [12]. In the context of password recovery, we describe the Hellman algorithm as follows.

We let X be the plaintext password and Y be the corresponding stored hash value of X . Given Y , we need to find X , which satisfies $h(X) = Y$, where h is a known hash function. However, finding $X = h^{-1}(Y)$ is feasibly impossible since hashes are computed using one-way functions, where the reversal function, h^{-1} , is unknown. Hellman suggested taking the plaintext values and applying alternate hashing and reducing, to generate a pre-computed table.

For example, the corresponding 128-bit hash value for a 7-character password (composed from a character set of English alphabets), is obtained by performing the password hashing function on the password. With a reduction function such as $H \bmod 26^7$, where H is the hash value converted to its decimal form, the resulting values are distributed in a best-effort uniform manner. For example, if we start with the initial plaintext value of "abcdefg" and upon hashing, we get a binary output of 0000000....000010000000....01, which is 64 '0's and a '1' followed by 62 '0's and a '1'. $H = 2^{63} + 1 = 9223372036854775809$. The reduction function will then convert this value to "3665127553", which corresponds to a plaintext representation "lwmkgij", computed from $(11(26^6) + 22(26^5) + 12(26^4) + 10(25^3) + 6(26^2) + 8(26^1) + 9(26^0))$. After a pre-defined number of rounds of hashing and reducing (making up a chain), only the initial and final plaintext values are stored. Therefore, only the "head" and "tail" of a chain are stored in the table. Using different initial plaintexts, the hashing and reducing operations are repeated, to generate a larger table (of increasing rows or chains). A larger table will theoretically contain more pre-computed values (i.e., disregarding hash collisions), thereby increasing the success rate of password recovery, while taking up more storage space. The pre-defined number of rounds of hashing and reducing will also increase the success rate by increasing the length of the "virtual" chain, while bringing about a higher computational overhead.

To recover a plaintext from a given hash, a reduction

operation is performed on the hash and a search for a match of the computed plaintext with the final value in the table is conducted. If a match is not found, the hashing and reducing operations are performed on the computed plaintext to arrive at a new plaintext so that another round of search to be made. The maximum number of rounds of hashing, reducing and searching operations is determined by the chain length. If the hash value is found in a particular chain, the values in the chain are then worked out by performing the hashing and reducing functions to arrive at the plaintext giving the specific hash value. Unfortunately, there is a likelihood that chains with different initial values may merge due to collisions. These merges will reduce the number of distinct hash values in the chains and therefore, diminish the rate of successful recovery. The success rate can be increased by using multiple tables with each table using a different reduction function. If we let $P(t)$ be the success rate of using t tables, then $P(t) = 1 - (1 - P(1))^t$, which is an increasing function of t since $P(1)$ is between 0 and 1. Hence, introducing more tables increase the success rate but also cause an increase in both the computational complexity and storage space.

In [19], Rivest suggested a method of using distinguished points as end points for chains. Distinguished points are keys, which satisfy a given criteria, e.g., the first or last q bits are all 0. In this method, the chains are not generated with a fixed length but they terminate upon reaching pre-defined distinguished points. This method decreases the number of memory lookups compared to Hellman's method and is capable of loop detection. If a distinguished point is not obtained after a large finite number of operations, the chain is suspected to contain a loop and is discarded. Therefore, the generated chains are free of loops. One limitation is that the chains will merge if there is a collision within the same table. The variable lengths of the chains will also result in an increase in the number of false alarms. Additional computations are also required to determine if a false alarm has occurred.

In 2003, Oechslin proposed a new table structure [17] to reduce the probability of merging occurrences. These rainbow chains use multiple reduction functions such that there will only be merges if the collisions occur at the same positions in both chains. An experiment was carried out and presented in Oechslin's paper. It showed that given a set of parameters, which is constant in both scenarios, the measured coverage in a single rainbow table is 78.8% compared to the 75.8% from the classical tables of Hellman with distinguished points. In addition, the number of calculations needed to perform the search is reduced as well.

In the following sections, we present our enhanced rainbow table [2] with a novel sorting algorithm [1], and propose 2 new storage methods, so that password lookup in the stored tables can be optimized.

III. ENHANCED RAINBOW TABLE

In this section, we present a new design of a time-memory trade-off precomputed table structure.

In this design, the same reduction functions as in the rainbow table method are used. The novelty lies in the

chains generation technique. Instead of taking a large set of plaintexts as our initial values, we systematically choose a much smaller unique set. We choose a plaintext and compute its corresponding hash value by applying the password hash algorithm. We let the resulting hash value written in decimal digits be H . Following that, we compute $(H+1) \bmod 2^j$, $(H+2) \bmod 2^j$, ..., $(H+k) \bmod 2^j$ for a variable k , where j is the number of bits of the hash output value. For example, in MD5 hash, $j = 128$. These hash values are then noted as the branches of the above chosen initial plaintext. We then proceed by applying alternate hashing and reducing operations to all these branches. We call this resulting extended chain, a block. The final values of the plaintexts are then stored with this 1 initial plaintext value. We perform the same operations for the other plaintexts. These sets of initial and final values make up the new pre-computed table.

To recover a password given a hash, we apply reducing and hashing operations alternatively until we obtain a plaintext that corresponds to one of the stored final values, as in the rainbow table method. After which, we generate the corresponding branch (e.g., if $k = 99$ and $chain\ id = 212$, the initial value is the initial plaintext in the third block and the branch id is 12), till the value of the password hash is reached.

A. Differences and Similarities in the designs

We identify and list the differences and similarities between the design of our new method and the rainbow table method as follow:

- Both use n reduction functions.
- Instead of storing the initial and final values as a pair as in the rainbow table, the initial value is stored with multiple output plaintexts after a series of hashing and reducing operations. This results in a large amount of storage conservation in the new method.
- The hashes H , $(H+1) \bmod 2^j$, $(H+2) \bmod 2^j$, ..., $(H+k) \bmod 2^j$ are calculated in order to generate subsequent hashes, resulting in the uniqueness of the values in the 1st column of hashes in the new method. The uniqueness of the hash values is guaranteed unless the total number of hashes is greater than 2^j . This situation is not likely to happen as it assumes an extremely large table, which fully stores all the possible pre-computed values.
- In this new design, the recovery of some passwords in the 1st column is not possible as they are not stored in the first place. However, we have shown in our analysis and evaluation [2] that the effect is negligible.

IV. SORTING ALGORITHM

The main drawback of the proposed enhanced rainbow table is that each password search will incur a significant amount of time complexity. The reason is that the passwords cannot be sorted in the usual alphabetical order now, since in doing so, the information of its corresponding initial hash value will be lost. The lookup will then have to rely on checking every single stored password in the table. Therefore, we propose a sorting algorithm so that password lookup in the stored tables can be optimized.

We require a sorting of the “tail” passwords to achieve a fast lookup. Therefore, we introduce special characters that cannot be found on the keyboard (i.e., non-printable ASCII characters). There are altogether 161 such characters and we assume that these non-printable ASCII characters do not form any of the character set of the passwords. We insert a number of these special characters into the passwords that we store. The manner in which these special characters are inserted will provide the information on the original position of the passwords in the rainbow table after the table has been rearranged in alphabetical order. The consequence is that this will incur more storage space but we will illustrate later that the increase in storage space is minimal and is also lesser than the original rainbow table’s storage requirement. The advantage of this sorting algorithm is that the passwords can now be sorted and thus a password lookup can be optimized.

A. Algorithm Design

Definition of notations:

Y = total number of special characters available

w = number of special characters in password

m = length of password

x = special character in password (labelled x_1 , x_2 , ...), $1 \leq x_i \leq Y$

p = location of a special character within password (labelled p_0 , p_1 , ...), $0 \leq p_i \leq m$, where x_i is placed at location p_{i-1} within a password

$${n \choose r} = \frac{n!}{(n-r)!r!}$$

From here on, position refers to the original position of a password in the rainbow table, while placement or location of a special character refers to its location in a password.

Password Position Computation

As an example, let 0000000 denote a 7-character password. The 161 non-printable ASCII characters are used as special characters x_1 , x_2 , ..., x_{161} , and are represented by numeric values from 1 to 161, respectively. The 8 possible locations of the special characters in a password are represented as underlines in _0_0_0_0_0_0_. Each location can hold more than one special character.

For example, 0000000 does not carry any special character and is at position 0. In 0000000x₁, x₁ is the first (scanning from rightmost) special character in the password, as denoted by its subscript value of 1. The position of this password depends on the numeric value represented by the special character, x₁. Therefore, the position is from 1 to 161 depending on which of the 161 special characters is used (i.e., 0000000x₁ is at position 1 if x₁ = 1, and at position 161 if x₁ = 161). Continuing in this manner, 000000x₁ is at position 162 if x₁ = 1, and at position 322 if x₁ = 161. Therefore, x₁0000000 is at position 1128 if x₁ = 1, and at position 1288 if x₁ = 161.

After covering all the locations for special characters in the password by using only 1 character but without completing

the allocation of special characters for all the passwords in the password space, we can increase the number of allocated special characters in the password, one at a time. For the insertion of 2 characters, 0000000x₂x₁ is at position 1289 if x₁=x₂=1. 0000000x₂x₁ is at position 1290 if x₂ = 1 and x₁ = 2. 0000000x₂x₁ is at position 1291 if x₂ = 1 and x₁ = 3. Continuing in this manner, 0000000x₂x₁ is at position 27209 if x₂ = x₁ = 161, 0000000x₂0x₁ is at position 27210 if x₂ = x₁ = 1, and so on.

We derive the following formulas for the computation of the original position of a password in the rainbow table.

w=1: Original position of password

$$Yp_0 + x_1$$

w=2: Original position of password

$$Yx_2 + x_1 + Y^2 \binom{p_1+1}{2} + Y^2 p_0 + Ym$$

w=3: Original position of password

$$Y^2 x_3 + Yx_2 + x_1 + Y^3 \binom{p_2+2}{3} + Y^3 \binom{p_1+1}{2} + Y^3 p_0 + Y^2 \binom{m+1}{2} + Y^2 m + Ym$$

w=4: Original position of password

$$Y^3 x_4 + Y^2 x_3 + Yx_2 + x_1 + Y^4 \binom{p_3+3}{4} + Y^4 \binom{p_2+2}{3} + Y^4 \binom{p_1+1}{2} + Y^4 p_0 + Y^3 \binom{m+2}{3} + Y^3 \binom{m+1}{2} + Y^3 m + Y^2 \binom{m+1}{2} + Y^2 m + Ym$$

For a general w, the original position is given by

$$\sum_{i=0}^{w-1} (Y^i x_{i+1} + Y^w \binom{p_i+i}{i+1}) + \sum_{i=0}^{w-2} \sum_{j=0}^i Y^{i+1} \binom{m+j}{j+1}$$

V. ANALYSIS

In this section, we present an analysis of the proposed enhanced rainbow table and its sorting algorithm. First, we analyse the maximum number of special characters required to sort tables of different sizes and password lengths, as well as demonstrate the storage conservation achieved. Next, we analyse the improvement in success rate of password recovery in the event of storage limitation. The 2 new storage methods are proposed and the impact on the storage conservation and success rate are demonstrated in this section.

A. Storage Conservation Analysis

Number of positions that can be assigned without using any special character

$$= 1$$

Number of positions that can be assigned using 1 special character

$$= Y(m+1)$$

Number of positions that can be assigned using 2 special characters

$$= Y^2[m+1+ \binom{m+1}{2}]$$

Number of positions that can be assigned using 3 special characters

$$= Y^3[m+1+ 2\binom{m+1}{2} + \binom{m+1}{3}]$$

Number of positions that can be assigned using 4 special characters

$$= Y^4[m+1+ 3\binom{m+1}{2} + 3\binom{m+1}{3} + \binom{m+1}{4}]$$

For w≥1, the number of positions that can be assigned using exactly w special characters

$$= Y^w \sum_{i=0}^{w-1} \binom{w-1}{i} \binom{m+1}{i+1}$$

Total number of positions that can be identified using at most w special characters (inclusive of positions that can be identified for number of special characters smaller than w) = $\sum_{i=0}^w \sum_{j=0}^i Y^i \binom{m+j-1}{j}$

Table I shows the number of positions that can be assigned given a pre-defined Y, m and w.

TABLE I: Total Number of Positions in Enhanced Rainbow Table

Y	m	w	Total Number of Positions
161	7	1	1,289
161	7	2	934,445
161	7	3	501,728,165
161	7	4	222,228,147,695
161	7	5	85,897,316,654,087
161	7	6	29,972,224,023,967,164
161	8	1	1,450
161	8	2	1,167,895
161	8	3	689,759,260
161	8	4	333,279,388,555
161	8	5	139,555,298,211,442
161	8	6	52,440,627,036,009,328
161	9	1	1,611
161	9	2	1,427,266
161	9	3	919,549,086
161	9	4	481,326,791,401
161	9	5	217,048,911,627,003
161	9	6	87,385,501,807,956,800
161	10	1	1,772
161	10	2	1,712,558
161	10	3	1,195,270,924
161	10	4	673,765,410,165
161	10	5	325,525,142,663,568
161	10	6	139,795,049,776,791,264

Let s be the total number of passwords to be stored in a rainbow table. Therefore, the total storage space required by the original rainbow table is (2 * m * s) bytes. In the enhanced rainbow table method, for s < $\sum_{i=0}^w \sum_{j=0}^i Y^i \binom{m+j-1}{j}$, at most w special characters are needed to be inserted to each password to identify its position. However, due to the use of special characters, the passwords to be stored are no longer of constant length. We propose two new methods to the storage of the passwords with their inserted special characters. In method 1, the passwords are still stored side by side as in the original rainbow table method. Retrieval of passwords for checking is performed at a specific fixed length. The w used in method 1 must be a fixed length too. The total number of positions that can be identified is reduced as they do not include positions that can

be identified for number of special characters smaller than w. The total number of positions in storage method 1 is shown in Table II.

TABLE II: Total Number of Positions in Storage Method 1 (side by side)

Y	m	w	Total Number of Positions
161	7	1	1,288
161	7	2	933,156
161	7	3	500,793,720
161	7	4	221,726,419,530
161	7	5	85,675,088,506,392
161	7	6	29,886,326,707,313,076
161	8	1	1,449
161	8	2	1,166,445
161	8	3	688,591,365
161	8	4	332,589,629,295
161	8	5	139,222,018,822,887
161	8	6	52,301,071,737,797,880
161	9	1	1,610
161	9	2	1,425,655
161	9	3	918,121,820
161	9	4	480,407,242,315
161	9	5	216,567,584,835,602
161	9	6	87,168,452,896,329,808
161	10	1	1,771
161	10	2	1,710,786
161	10	3	1,193,558,366
161	10	4	672,570,139,241
161	10	5	324,851,377,253,403
161	10	6	139,469,524,634,127,680

In storage method 2, we propose storing the passwords line by line. Therefore, a special character has to be used for delimitation purpose. A good choice would be the line feed character. Y is then reduced to 160. The w used in method 2 can be of a variable length. The total number of positions that can be identified still includes positions that can be identified for number of special characters smaller than w. The total number of positions in storage method 2 is shown in Table III.

In storage method 1, the total storage space needed = $(m+w)s$

In storage method 2, the total storage space needed
 $= \sum_{i=1}^{w-1} \sum_{j=0}^{i-1} Y^i (m+i+1) \binom{i-1}{j} \binom{m+1}{j+1} + (m+w+1)[s + 1 - \sum_{i=0}^{w-1} \sum_{j=0}^i Y^i \binom{m+j-1}{j}]$

We consider two main scenarios in the performance evaluation based on storage space. In the first scenario, any alpha-numeric characters can be used in the passwords. There would be 62 characters in total. In the second scenario, we increase the password character set to include all printable ASCII characters, which will consist of 95 characters.

TABLE III: Total Number of Positions in Storage Method 2 (line by line)

Y	m	w	Total Number of Positions
160	7	1	1,281
160	7	2	922,881
160	7	3	492,442,881
160	7	4	216,761,242,881
160	7	5	83,263,980,442,881
160	7	6	28,872,966,636,442,880
160	8	1	1,441
160	8	2	1,153,441
160	8	3	676,993,441
160	8	4	325,080,193,441
160	8	5	135,276,811,393,441
160	8	6	50,517,256,459,393,440
160	9	1	1,601
160	9	2	1,409,601
160	9	3	902,529,601
160	9	4	469,484,929,601
160	9	5	210,394,400,129,601
160	9	6	84,180,360,480,129,600
160	10	1	1,761
160	10	2	1,691,361
160	10	3	1,173,147,361
160	10	4	657,188,507,361
160	10	5	315,544,561,307,361
160	10	6	134,667,490,289,307,360

Scenario 1: Alpha-numeric character set in passwords

We consider the cases where the passwords are 7, 8, 9 and 10 characters in length. Table IV shows the required number of special characters, w, to store all the passwords when the number of hashing and reduction functions (i.e., virtual columns) in the rainbow table are 30,000.

TABLE IV: Required w for Alpha-Numeric Passwords in Both Storage Methods when virtual columns are 30,000

Password Length	Total Password Space (s)	w in Method 1	w in Method 2
7	117,387,154	3	3
8	7,278,003,520	4	4
9	451,236,218,209	4	4
10	27,976,645,528,945	5	5

Table V shows the required number of special characters, w, to store all the passwords when the virtual columns in the rainbow table are 100,000.

Table VI shows the storage requirement to store all the passwords when the virtual columns in the rainbow table are 30,000, while Table VII shows the storage requirement when the virtual columns are 100,000. In the case when the number of virtual columns in the rainbow table is set to 30,000, the improvement in terms of storage conservation of method 1 over the original rainbow table was 28.57%,

TABLE V: Required w for Alpha-Numeric Passwords in Both Storage Methods when virtual columns are 100,000

Password Length	Total Password Space (s)	w in Method 1	w in Method 2
7	35,216,147	3	3
8	2,183,401,056	4	4
9	135,370,865,463	4	4
10	8,392,993,658,684	5	5

25%, 27.78%, and 25% for password length of 7, 8, 9 and 10, respectively. Comparing method 1 over method 2, the improvement in terms of storage conservation was 9.03%, 7.03%, 7.13%, and 6.11% for password length of 7, 8, 9 and 10, respectively.

In the case when the number of virtual columns in the rainbow table is set to 100,000, the improvement in terms of storage conservation of method 1 over the original rainbow table was 28.57%, 25%, 27.78%, and 25% for password length of 7, 8, 9 and 10, respectively. Comparing method 1 over method 2, the improvement in terms of storage conservation was 8.87%, 5.43%, 7.10%, and 5.79% for password length of 7, 8, 9 and 10, respectively.

TABLE VI: Storage Requirement for Alpha-Numeric Passwords when virtual columns is 30,000

Password Length	Original Rainbow Table	Method 1	Method 2
7	1,643,420,156B 1.53GB 0.0015TB	1,173,871,540B 1.09GB 0.0011TB	1,290,334,534B 1.20GB 0.0012TB
8	116,448, 056,320B 108,454B 0.1059TB	87,336, 042,240B 81,34GB 0.0794TB	93,935, 897,440B 87,48GB 0.0854TB
9	8,122,251, 927,762B 7,564,44GB 7.39TB	5,866, 070,836,717B 5,463,20GB 5.34TB	6,316,403, 114,126B 5,882,61GB 5.74TB
10	559,532,910, 578,900B 521,105,63GB 508.89TB	419,649,682, 934,175B 390,829,22GB 381.67TB	446,967,965, 115,280B 416,271.36GB 406.51TB

Scenario 2: All printable ASCII character set in passwords

We consider the cases where the passwords are 7, 8, and 9 characters in length. Table VIII shows the required number of special characters, w, to store all the passwords when the number of hashing and reduction functions (i.e., virtual columns) in the rainbow table are 30,000.

Table IX shows the required number of special characters, w, to store all the passwords when the virtual columns in the rainbow table are 100,000.

TABLE VII: Storage Requirement for Alpha-Numeric Passwords when virtual columns is 100,000

Password Length	Original Rainbow Table	Method 1	Method 2
7	493,026,058B 0.46GB 0.00045TB	352,161,470B 0.33GB 0.00032TB	386,453,457B 0.36GB 0.00035TB
8	34,934, 416,896B 32.54GB 0.0318TB	26,200, 812,672B 24.40GB 0.0238TB	27,706, 865,408B 25.80GB 0.0252TB
9	2,436,675, 578,334B 2,269.33GB 2.22TB	1,759,821, 251,019B 1,638.96GB 1.60TB	1,894,288, 175,682B 1764.19GB 1.72TB
10	167,859,873, 173,680B 156,331.69GB 152.67TB	125,894,904, 880,260B 117,248.77GB 114.50TB	133,629,535, 191,104B 124,452.20GB 121.54TB

TABLE VIII: Required w for All Printable ASCII Character Passwords in Both Storage Methods when virtual columns are 30,000

Password Length	Total Password Space (s)	w in Method 1	w in Method 2
7	2,327,790,987	4	4
8	221,140,143,764	4	4
9	21,008,313,657,487	5	5

TABLE IX: Required w for All Printable ASCII Character Passwords in Both Storage Methods when virtual columns are 100,000

Password Length	Total Password Space (s)	w in Method 1	w in Method 2
7	698,337,297	4	4
8	66,342,043,129	4	4
9	6,302,494,097,247	5	5

Table X shows the storage requirement to store all the passwords when the virtual columns in the rainbow table are 30,000, while Table XI shows the storage requirement when the virtual columns are 100,000. In the case when the number of virtual columns in the rainbow table is set to 30,000, the improvement in terms of storage conservation of method 1 over the original rainbow table was 21.43%, 25%, and 22.22% for password length of 7, 8 and 9, respectively. Comparing method 1 over method 2, the improvement in terms of storage conservation was 6.69%, 7.67%, and 6.53% for password length of 7, 8 and 9, respectively.

In the case when the number of virtual columns in the rainbow table is set to 100,000, the improvement in terms of storage conservation of method 1 over the original rainbow table was 21.43%, 25%, and 22.22% for password length of 7, 8 and 9, respectively. Comparing method 1 over method

2, the improvement in terms of storage conservation was 2.60%, 7.62%, and 6.20% for password length of 7, 8 and 9, respectively.

TABLE X: Storage Requirement for All Printable ASCII Character Passwords when virtual columns is 30,000

Password Length	Original Rainbow Table	Method 1	Method 2
7	32,589, 073,818B 30.35GB 0.0296TB	25,605, 700,857B 23.85GB 0.0233TB	27,440, 124,804B 25.56GB 0.0250TB
8	3,538,242, 300,224B 3,295.24GB 3.22TB	2,653,681, 725,168B 2,471.43GB 2.41TB	2,874,143, 720,612B 2,676.75GB 2.61TB
9	378,149,645, 834,766B 352,179.30GB 343.93TB	294,116,391, 204,818B 273,917.23GB 267.50TB	314,654,315, 991,905B 293,044.67GB 286.18TB

TABLE XI: Storage Requirement for All Printable ASCII Character Passwords when virtual columns is 100,000

Password Length	Original Rainbow Table	Method 1	Method 2
7	9,776, 722,158B 9.11GB 0.0089TB	7,681, 710,267B 7.15GB 0.0070TB	7,886, 680,524B 7.35GB 0.0072TB
8	1,061,472, 690,064B 988.57GB 0.9654TB	796,104, 517,548B 741.43GB 0.7241TB	861,768, 412,357B 802.58GB 0.7838TB
9	113,444,893, 750,446B 105,653.79GB 103.18TB	88,234,917, 361,458B 82,175.17GB 80.25TB	94,067,022, 588,305B 87,606.74GB 85.55TB

B. Success Rate Improvement Analysis

Here, we analyse the improvement in terms of success rate of password recovery. To do so, we set the storage requirement to be a fixed value and compute the achievable success rate. Table XII to Table XV shows the success rate when the storage is capped at a certain value and the virtual columns are 30,000 for different password lengths, while Table XVI to Table XIX shows the evaluation when the virtual columns are 100,000 instead. The password character set consists of the alpha-numeric characters.

TABLE XII: Success Rate for 7-Character Alpha-Numeric Passwords when virtual columns are 30,000

Storage Limit	Original Rainbow Table	Method 1	Method 2
0.5GB	$\frac{38347922}{117387154} = 32.67\%$	$\frac{53687091}{117387154} = 45.74\%$	$\frac{48890461}{117387154} = 41.65\%$
1GB	$\frac{76695844}{117387154} = 65.34\%$	$\frac{107374182}{117387154} = 91.47\%$	$\frac{97696907}{117387154} = 83.23\%$
1.5GB	$\frac{115043766}{117387154} = 98\%$	$\frac{117387154}{117387154} = 100\%$	$\frac{117387154}{117387154} = 100\%$

TABLE XIII: Success Rate for 8-Character Alpha-Numeric Passwords when virtual columns are 30,000

Storage Limit	Original Rainbow Table	Method 1	Method 2
50GB	$\frac{3355443200}{7278003520} = 46.10\%$	$\frac{4473924266}{7278003520} = 61.47\%$	$\frac{4181941501}{7278003520} = 57.46\%$
75GB	$\frac{5033164800}{7278003520} = 69.16\%$	$\frac{6710886400}{7278003520} = 92.21\%$	$\frac{6246829624}{7278003520} = 85.83\%$
100GB	$\frac{6710886400}{7278003520} = 92.21\%$	$\frac{7278003520}{7278003520} = 100\%$	$\frac{7278003520}{7278003520} = 100\%$

We observe from the evaluations that in the event of storage limitation, method 1 performs significantly better in

TABLE XIV: Success Rate for 9-Character Alpha-Numeric Passwords when virtual columns are 30,000

Storage Limit	Original Rainbow Table	Method 1	Method 2
1TB	$\frac{61083979320}{451236218209} = 13.54\%$	$\frac{84577817521}{451236218209} = 18.74\%$	$\frac{78601112041}{451236218209} = 17.42\%$
3TB	$\frac{183251937962}{451236218209} = 40.61\%$	$\frac{253733452563}{451236218209} = 56.23\%$	$\frac{235674201723}{451236218209} = 52.23\%$
5TB	$\frac{305419896604}{451236218209} = 67.69\%$	$\frac{422889087606}{451236218209} = 93.72\%$	$\frac{392747291405}{451236218209} = 87.04\%$

TABLE XV: Success Rate for 10-Character Alpha-Numeric Passwords when virtual columns are 30,000

Storage Limit	Original Rainbow Table	Method 1	Method 2
100TB	$\frac{5497558138880}{27976645528945} = 19.65\%$	$\frac{7330077518506}{27976645528945} = 26.20\%$	$\frac{6913095382840}{27976645528945} = 24.71\%$
300TB	$\frac{16492674416640}{27976645528945} = 58.95\%$	$\frac{2199023255520}{27976645528945} = 78.60\%$	$\frac{20656990730040}{27976645528945} = 73.84\%$
500TB	$\frac{27487790694400}{27976645528945} = 98.25\%$	$\frac{27976645528945}{27976645528945} = 100\%$	$\frac{27976645528945}{27976645528945} = 100\%$

TABLE XVI: Success Rate for 7-Character Alpha-Numeric Passwords when virtual columns are 100,000

Storage Limit	Original Rainbow Table	Method 1	Method 2
0.1GB	$\frac{7669584}{35216147} = 21.78\%$	$\frac{10737418}{35216147} = 30.49\%$	$\frac{9845303}{35216147} = 27.96\%$
0.2GB	$\frac{15339168}{35216147} = 43.56\%$	$\frac{21474836}{35216147} = 60.98\%$	$\frac{19606593}{35216147} = 55.68\%$
0.3GB	$\frac{23008753}{35216147} = 65.34\%$	$\frac{32212254}{35216147} = 91.47\%$	$\frac{29367882}{35216147} = 83.39\%$

terms of password recovery success rate compared to both the original rainbow table and method 2, consistently. Based on the results above, the improvement in success rate of method 1 over the original rainbow table can reach up to 26.13%.

Table XX to Table XXII shows the success rate when the storage is capped at a certain value and the virtual columns are 30,000 for different password lengths, while Table XXIII to Table XXV shows the evaluation when the virtual columns are 100,000 instead. The password character set consists of all the printable ASCII characters.

TABLE XVII: Success Rate for 8-Character Alpha-Numeric Passwords when virtual columns are 100,000

Storage Limit	Original Rainbow Table	Method 1	Method 2
10GB	$\frac{671088640}{2183401056} = 30.74\%$	$\frac{894784853}{2183401056} = 40.98\%$	$\frac{878120504}{2183401056} = 40.22\%$
20GB	$\frac{1342177280}{2183401056} = 61.47\%$	$\frac{1789569706}{2183401056} = 81.96\%$	$\frac{1704075753}{2183401056} = 78.05\%$
30GB	$\frac{2013265920}{2183401056} = 92.21\%$	$\frac{2183401056}{2183401056} = 100\%$	$\frac{2183401056}{2183401056} = 100\%$

TABLE XVIII: Success Rate for 9-Character Alpha-Numeric Passwords when virtual columns are 100,000

Storage Limit	Original Rainbow Table	Method 1	Method 2
1TB	$\frac{61083979320}{135370865463} = 45.12\%$	$\frac{84577817521}{135370865463} = 62.48\%$	$\frac{78601112041}{135370865463} = 58.06\%$
1.5TB	$\frac{91625968981}{135370865463} = 67.69\%$	$\frac{126866726281}{135370865463} = 93.72\%$	$\frac{117869384461}{135370865463} = 87.07\%$
2TB	$\frac{122167958641}{135370865463} = 90.25\%$	$\frac{135370865463}{135370865463} = 100\%$	$\frac{135370865463}{135370865463} = 100\%$

TABLE XIX: Success Rate for 10-Character Alpha-Numeric Passwords when virtual columns are 100,000

Storage Limit	Original Rainbow Table	Method 1	Method 2
50TB	$\frac{2748779069440}{8392993658684} = 32.75\%$	$\frac{3665038759253}{8392993658684} = 43.67\%$	$\frac{3477121546040}{8392993658684} = 41.43\%$
100TB	$\frac{5497558138880}{8392993658684} = 65.50\%$	$\frac{7330077518506}{8392993658684} = 87.34\%$	$\frac{6913095382840}{8392993658684} = 82.37\%$
150TB	$\frac{8246337208320}{8392993658684} = 98.25\%$	$\frac{8392993658684}{8392993658684} = 100\%$	$\frac{8392993658684}{8392993658684} = 100\%$

TABLE XX: Success Rate for 7-Character Printable ASCII Passwords when virtual columns are 30,000

Storage Limit	Original Rainbow Table	Method 1	Method 2
10GB	$\frac{766958445}{2327790987} = 32.95\%$	$\frac{976128930}{2327790987} = 41.93\%$	$\frac{935898773}{2327790987} = 40.21\%$
20GB	$\frac{1533916891}{2327790987} = 65.90\%$	$\frac{1952257861}{2327790987} = 83.87\%$	$\frac{1830683626}{2327790987} = 78.64\%$
30GB	$\frac{2300875337}{2327790987} = 98.84\%$	$\frac{2327790987}{2327790987} = 100\%$	$\frac{2327790987}{2327790987} = 100\%$

TABLE XXI: Success Rate for 8-Character Printable ASCII Passwords when virtual columns are 30,000

Storage Limit	Original Rainbow Table	Method 1	Method 2
1TB	$\frac{68719476736}{221140143764} = 31.08\%$	$\frac{91625968981}{221140143764} = 41.43\%$	$\frac{84629982776}{221140143764} = 38.27\%$
2TB	$\frac{137438953472}{221140143764} = 62.15\%$	$\frac{183251937962}{221140143764} = 82.87\%$	$\frac{169207800297}{221140143764} = 76.52\%$
3TB	$\frac{206158430208}{221140143764} = 93.23\%$	$\frac{221140143764}{221140143764} = 100\%$	$\frac{221140143764}{221140143764} = 100\%$

TABLE XXII: Success Rate for 9-Character Printable ASCII Passwords when virtual columns are 30,000

Storage Limit	Original Rainbow Table	Method 1	Method 2
100TB	$\frac{6108397932088}{21008313657487} = 29.08\%$	$\frac{7853654484114}{21008313657487} = 37.38\%$	$\frac{7361436776533}{21008313657487} = 35.04\%$
200TB	$\frac{12216795864177}{21008313657487} = 58.15\%$	$\frac{15707308968228}{21008313657487} = 74.77\%$	$\frac{14691514295040}{21008313657487} = 69.93\%$
300TB	$\frac{18325193796266}{21008313657487} = 87.23\%$	$\frac{21008313657487}{21008313657487} = 100\%$	$\frac{21008313657487}{21008313657487} = 100\%$

Similarly, in the case of using all printable ASCII characters as the password character set, we observe from the evaluations that in the event of storage limitation, method 1 performs significantly better in terms of password recovery success rate compared to both the original rainbow table and method 2, consistently. Based on the results above, the improvement in success rate of method 1 over the original rainbow table can reach up to 23.60%.

TABLE XXIII: Success Rate for 7-Character Printable ASCII Passwords when virtual columns are 100,000

Storage Limit	Original Rainbow Table	Method 1	Method 2
3GB	$\frac{230087533}{698337297} = 32.95\%$	$\frac{292838679}{698337297} = 41.93\%$	$\frac{309549376}{698337297} = 27.96\%$
5GB	$\frac{383479222}{698337297} = 54.91\%$	$\frac{488064465}{698337297} = 69.89\%$	$\frac{488506346}{698337297} = 69.95\%$
7GB	$\frac{536870912}{698337297} = 76.88\%$	$\frac{683290251}{698337297} = 97.85\%$	$\frac{667463317}{698337297} = 95.58\%$

TABLE XXIV: Success Rate for 8-Character Printable ASCII Passwords when virtual columns are 100,000

Storage Limit	Original Rainbow Table	Method 1	Method 2
500GB	$\frac{33554432000}{66342043129} = 50.58\%$	$\frac{44739242666}{66342043129} = 67.44\%$	$\frac{41349927716}{66342043129} = 62.33\%$
700GB	$\frac{46976204800}{66342043129} = 70.81\%$	$\frac{62634939733}{66342043129} = 94.41\%$	$\frac{57869032701}{66342043129} = 87.23\%$
900GB	$\frac{60397977600}{66342043129} = 91.04\%$	$\frac{66342043129}{66342043129} = 100\%$	$\frac{66342043129}{66342043129} = 100\%$

TABLE XXV: Success Rate for 9-Character Printable ASCII Passwords when virtual columns are 100,000

Storage Limit	Original Rainbow Table	Method 1	Method 2
50TB	$\frac{3054198966044}{6302494097247} = 48.46\%$	$\frac{3926827242057}{6302494097247} = 62.31\%$	$\frac{3696398017280}{6302494097247} = 58.65\%$
75TB	$\frac{4581298449066}{6302494097247} = 72.69\%$	$\frac{5890240863085}{6302494097247} = 93.46\%$	$\frac{5528917396906}{6302494097247} = 87.73\%$
100TB	$\frac{6108397932088}{6302494097247} = 96.92\%$	$\frac{6302494097247}{6302494097247} = 100\%$	$\frac{6302494097247}{6302494097247} = 100\%$

VI. CONCLUSIONS

This paper briefly describes our previous work on an enhanced rainbow table design [2] coupled with a sorting algorithm [1], which when applied, has a significant improvement over the orginal rainbow tables. Special characters are added to the storage to allow the sorting of the enhanced rainbow tables so that the password lookup time can be optimized. We further proposed 2 new storage methods to be applied with the enhanced rainbow table and showed that even with this insertion of characters to the passwords, the improvement in storage space required to store the same number of passwords reaches 28.57% lesser than what is required in the original tables in the case of alpha-numeric character set. The improvement, when the password character set consists of all the printable ASCII characters, reaches 25%. This is achieved while maintaining the same success rate.

By considering storage space limitations, we also evaluated the achieveable success rate of password recovery in different scenarios. Our analysis shows that an improvement of up to 26.13% and 23.60% can be achieved in terms of success rate, when compared to the original rainbow tables, for the alpha-numeric passwords and passwords containing any of the printable ASCII characters.

References

- [1] H. M. Ying and V. L. L. Thing, "A novel rainbow table sorting method", International Conference on Technical and Legal Aspects of the e-Society (CYBERLAWS), February 2011
- [2] V. L. L. Thing and H. M. Ying, "A novel time-memory trade-off method for password recovery", Digital Investigation, International Journal of Digital Forensics and Incident Response, Elsevier, Vol. 6, Supplement, pp. S114-S120, September 2009
- [3] S. M. Smyth, "Searches of computers and computer data at the United States border: The need for a new framework following United States V. Arnold", Journal of Law, Technology and Policy, Vol. 2009, No. 1, pp. 69-105, February 2009.
- [4] Google News, "Favorite passwords: '1234' and 'password'", <http://www.google.com/hostednews/afp/article/ALeqM5jeUc6Bblnd0M19WVQWvjS6D2puvw>, [retrieved, January 2012].
- [5] Cain and Abel, "Password recovery tool", <http://www.oxid.it>, [retrieved, January 2012].
- [6] John The Ripper, "Password cracker", <http://www.openwall.com>, [retrieved, January 2012].
- [7] LCPSOFT, "Lcpsoft programs", <http://www.lcpsoft.com>, [retrieved, January 2012].
- [8] S. Contini and Y. L. Yin, "Forgery and partial key-recovery attacks on HMAC and NMAC using hash collisions", Annual International Conference on the Theory and Application of Cryptology and Information Security (AsiaCrypt), Lecture Notes in Computer Science, Vol. 4284, pp. 37-53, 2006.
- [9] P. A. Fouque, G. Leurent, and P. Q. Nguyen, "Full key-recovery attacks on HMAC/NMAC-MD4 and NMAC-MD5", Advances in Cryptology, Lecture Notes in Computer Science, Vol. 4622, pp. 13-30, Springer, 2007.
- [10] Y. Sasaki, G. Yamamoto, and K. Aoki, "Practical password recovery on an MD5 challenge and response", Cryptology ePrint Archive, Report 2007/101, April 2008.
- [11] Y. Sasaki, L. Wang, K. Ohta, and N. Kunihiro, "Security of MD5 challenge and response: Extension of APOP password recovery attack", The Cryptographers' Track at the RSA Conference on Topics in Cryptology, Vol. 4964, pp. 1-18, April 2008.
- [12] M. E. Hellman, "A cryptanalytic time-memory trade-off", IEEE Transactions on Information Theory, Vol. IT-26, No. 4, pp. 401-406, July 1980.
- [13] D. Todorov, "Mechanics of user identification and authentication: Fundamentals of identity management", Auerbach Publications, Taylor and Francis Group, June 2007.
- [14] R. Rivest, "The MD5 message-digest algorithm", IETF RFC 1321, April 1992.
- [15] National Institute of Standards and Technology (NIST), "Secure hash standard", Federal Information Processing Standards Publication 180-2, August 2002.
- [16] H. Dobbertin, A. Bosselaers, and B. Preneel, "Ripemd-160: A strengthened version of RIPEMD", International Workshop on Fast Software Encryption, Lecture Notes in Computer Science, Vol. 1039, pp. 71-82, Springer, April 1996.
- [17] P. Oechslin, "Making a faster cryptanalytic time-memory trade-off", Annual International Cryptology Conference (CRYPTO), Advances in Cryptography, Lecture Notes in Computer Science, Vol. 279, pp. 617-630, October 2003.
- [18] A. Narayanan and V. Shmatikov, "Fast dictionary attacks on passwords using time-space tradeoff", ACM Conference on Computer and Communications Security, pp. 364-372, 2005.
- [19] D. E. R. Denning, "Cryptography and data security", Addison-Wesley Publication, 1982.

An Augmented Reality Platform for the Enhancement of Surgical Decisions in Pediatric Laparoscopy

Lucio Tommaso De Paolis, Giovanni Aloisio

Department of Innovation Engineering
University of Salento

Lecce, Italy

lucio.depaolis@unisalento.it
giovanni.aloisio@unisalento.it

Abstract— The practice of Minimally Invasive Surgery is becoming more and more widespread and adopted as an alternative to the classical procedure. This technique presents many advantages for the patients, but also some limitations for the surgeons. In particular, the lack of depth in perception and the difficulty in estimating the distance of the specific structures in laparoscopic surgery can impose limits on delicate dissection or suturing. The presence of new systems for the pre-operative planning can be of great help to the surgeon. The use of the Augmented Reality technology shows a way forward in bringing the direct advantage of the visualization of the open surgery back to minimally invasive surgery and can increase for the physician the view of the organs with information obtained from the image processing of the patient. The developed application allows the surgeon to get information about the patient and her/his pathology, visualizing and interacting with the 3D models of the organs built from the patient's medical images, measuring the dimensions of the organs and deciding the best insertion points of the trocars in the patient's body. This choice can be visualized on the real patient using the Augmented Reality technology.

Keywords - Augmented Reality; medical image processing; user interface; minimally invasive surgery; preoperative surgical planning

I. INTRODUCTION

One trend in surgery is the transition from open procedures to minimally invasive laparoscopic interventions, where visual feedback to the surgeon is only possible through the laparoscope camera and direct palpation of organs is not possible.

Minimally Invasive Surgery (MIS), such as laparoscopy or endoscopy, has become very important and the research in this field is more and more widely accepted. These techniques offer the possibility to surgeons of reaching the patient's internal anatomy in a less invasive way and causing only a minimal trauma to patients.

The diseased area is reached by means of small incisions in the body, called ports. Specific instruments and a camera are inserted through these ports; during the operation a monitor shows what is going on inside the body. The

surgeon does not have a direct vision of the organs and thus he is guided by camera images; this is very different from what happens in open surgery because there is no possibility to touch the organs.

The laparoscopic access is an alternative to the open entry techniques because it aims to prevent visceral and vascular injury due to division of abdominal wall layers. The reasons of a limited use of the open-access method is due to the time needed for the performance, the difficulty in maintaining the pneumoperitoneum because of the gas leakage and the lack of a particular evidence for the prevention of intra-abdominal injury using this method.

The vascular injury during the first laparoscopic access is the first cause of death in laparoscopy, second only to anesthesia and bowel injury, with a reported mortality rate of 15%.

Unlike most of vascular injuries, where the occurrence and presentation are immediate, many bowel injuries are not recognized at the time of the procedure because of the suboptimal visualization.

To overpass the several complications in the laparoscopic access, optically guided trocars are designed to decrease the risk of injury to intra-abdominal structures allowing the surgeon to visualize abdominal wall layers during the placement.

As a promising technique, the practice of MIS is becoming more and more widespread and is being adopted as an alternative to classical procedures.

Shorter hospitalizations, faster bowel function return, fewer wound-related complications and a more rapid return to normal activities have contributed to accept these surgical procedures.

The advantages of this surgical method are evident on the patients, but these techniques involve some limitations to surgeons; due to the limited field of view, the position and the orientation of the camera require frequent adjustments and significant hand-eye coordination is necessary because the instrument movements visualized on the screen not match the surgeon's hand movements.

In addition, the imagery is in 2D and the surgeon can

estimate the distance of anatomical structures only by moving the camera. In laparoscopic surgery, the lack of depth perception and the difficulty in estimating the distance from the anatomical structures can impose limitations on delicate dissection or suturing.

Motivated by the benefits that MIS can bring to patients, many research groups are now focusing on the development of systems in order to assist the surgeons during the surgical procedures and to carry out their tasks in both faster and safer ways. Other research groups have developed solutions to support the preoperative surgical planning and the intra-operative surgical procedure.

Even though the interpretation of the computed tomography (CT) or the magnetic resonance images (MRI) remains a difficult task, the latest developments in medical imaging processing make possible the reconstruction of 3D models of the organs providing anatomical information barely detectable by CT and MRI slices or ultrasound scan and an accurate knowledge of patient's anatomy and pathologies as well.

A suitable use of these models could lead to an improvement in patient care by guiding the instruments through the body without the direct sight of the physician; in addition, these models can be the bases to build the realistic virtual environment used in Virtual Reality and Augmented Reality applications.

This paper presents an advanced platform for the visualization and the interaction with the 3D patient models of the organs built from CT images [1].

The presence of a system for the pre-operative planning can help the surgeon very much and this support is more and more important in pediatric laparoscopic surgery where you have to understand the exact conditions of the patient's organs and the precise location of the operational site.

The developed application allows the surgeon to choose the points for the insertion of the trocars on the virtual model and to overlap them, before starting the real surgical procedure, on the real patient body using the Augmented Reality technology.

II. THE AUGMENTED REALITY IN SURGERY

Appropriate visualization tools and techniques play an important role in providing detailed information about human organs, pathologies and realistic 3D models of the organs of the specific patient. The utilization of visual information together with the operation techniques help the surgeon during the surgical procedure and provide a possible solution to the problems that minimally invasive surgery can present.

In addition, the integration with the Virtual Reality technology can change surgical preparation and the surgeons can practice and perform a surgical procedure before the patient arrives in the operating room; this

involves not only a reduction of complications, but also individual components of the surgery can be honed to precision.

The use of the Augmented Reality technology shows a way forward in bringing the direct advantage of the visualization of the open surgery back to minimally invasive surgery and can increase for the physician the view of the organs with information obtained from the image processing of the patient [2].

Augmented Reality can avoid some drawbacks of MIS and can lead to new medical treatments.

The Augmented Reality research aims to allow the real-time fusion between the computer-generated digital content and the real world. Thanks to Augmented Reality, it is possible to see hidden objects and therefore to enhance the users' perception and to improve the interaction with the real world. The virtual objects, displaying what the users cannot detect directly with their own senses, help them to perform real-world tasks better.

In opposition to Virtual Reality technology that gets into a synthetic environment but doesn't make possible the vision of the real world, Augmented Reality technology allows to see 3-dimensional virtual objects superimposed upon the real world.

Therefore, AR supplements reality rather than completely replace it. The user has a feeling that the virtual and real objects coexist in the same space.

Azuma [3] presents a survey of AR and describes the characteristics of AR systems, registration and sensing errors together with the efforts to overcome them. Using Azuma's definition, an AR system has to fulfil the following three characteristics:

- Real and virtual objects are united in a real environment, they appear to coexist in the same space;
- The system is interactive and it performs in real-time;
- The virtual objects are registered with the real world.

Milgram and Kishino [4] defined the Mixed Reality as an environment "in which real world and virtual world objects are presented together within a single display, that is, anywhere between the extrema of the virtuality continuum"

The Virtuality Continuum extends from the completely real through to the completely virtual environment with Augmented Reality and Augmented Virtuality ranging between.

Thus Augmented Reality is a mixture of reality and virtual reality. It includes both virtual objects and real-world elements, but the surrounding environment is real.

Fig. 1 shows the Milgram's reality-virtuality continuum.

In order to have a true AR application, the computer-generated organs must be accurately positioned on the real ones. For this reason it is necessary to carry out an accurate

registration phase, which provides, as result, the correct overlapping of the 3D model of the virtual organs on the real patient [5], [6], [7].

In medical applications of the Augmented Reality technology, the right detection and the overlapping of the fiducial points are very important because even a very slight mistake could have very serious consequences on the patient.

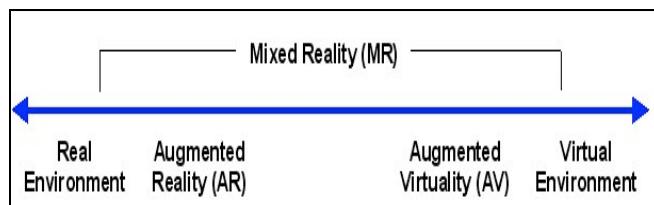


Figure 1. Milgram's reality-virtuality continuum

The integration of the registration algorithm into the surgical workflow requires a trade-off of complexity, accuracy and invasiveness. The process of registration can be obtained using the optical (infrared) tracking systems that are the best choice at the moment; these devices are already in use in the modern operating rooms.

For the registration of patient data with the AR system it is possible to have a point-based registration approach where specific fiducials can be used and fixed on the of the patient. These fiducials are touched with a tracked pointer and their positions have to match the correspondent positions of fiducials placed during the patient scanning and segmented in the 3D model. Point-based registration is known to be a reliable solution if the set of fiducials is carefully chosen. The accuracy depends on the number of fiducials, the quality of measurement and the spatial fiducial arrangement [8].

The simple augmentation of the real scene is not realistic enough because, although the organ positions are computed correctly, the relative position in depth of real and virtual images may not be perceived.

Indeed, in AR applications, although virtual objects have been correctly positioned in the scene, they are visually overlapped with all real objects, creating a situation that is not sufficiently realistic.

In particular, this effect is not acceptable for surgical AR applications and it is necessary, in addition to a proper positioning of the organs in the virtual scene, in order to ensure a correct visualization.

Some solutions have been proposed, but the issue of correct depth visualization remains partially unsolved.

Augmented Reality provides an intuitive human-computer interface. In surgery this technology makes it possible to overlay virtual medical images on the patient, allowing surgeons to have a sort of "X-ray vision" of the body and providing a view of the patient's anatomy.

Augmented Reality technology offers the same visual advantages as open surgery in minimally invasive surgery and increases the physician's visual knowledge with information gathered from patients' medical images.

The patient becomes transparent and this virtual transparency makes it possible to find tumours or vessels not using the touch, but simply visualizing them thanks to augmented reality.

The virtual information can be displayed directly on the patient's body or visualized on an AR surgical interface, showing where the operation must be performed.

For instance, a physician might also be able to see the exact location of a lesion on a patient's liver or the right place where to drill a hole on the skull in brain surgery or where to perform a needle biopsy of a tiny tumour.

In general, AR technology may be used in minimally invasive surgery for:

- Training purposes;
- Preoperative planning;
- Advanced visualization during the real procedure.

AR technology in minimally invasive surgery may be used for training purposes, pre-operative planning and advanced visualization during the real procedure. Several research groups are exploring the use of AR in surgery and are developing many image-guided surgery systems.

Devernay et al. [9] propose the use of an endoscopic AR system for robotically assisted minimally invasive cardiac surgery. One of the problems closely linked to endoscopic surgery is that, because of the narrow field of view, sometimes it is quite difficult to locate the objects seen through the endoscope. The information coming from the 3D anatomical model of the patient's organs (built from MRI or CT-scan) and the position of the endoscope are not sufficient because some organs are displaced by the inflated gas. They propose a methodology to achieve coronary localization by Augmented Reality on a robotized stereoscopic endoscope adding "cartographic" information on the endoscopic view and by indicating the position of the coronaries with respect to the field of view.

Bichlmeier et al. [10], [11] focus on the problem of misleading perception of depth and spatial layout in medical AR and present a new method for medical in-situ visualization that allows for improved perception of 3D medical imaging data and navigated surgical instruments relative to the patient's anatomy. They describe a technique to modify the transparency of video images recorded by the colour cameras of a video see-through HMD. The presented method allows for an intuitive view on the deep-seated anatomy of the patient providing visual cues to perceive correctly absolute and relative distances of objects within an AR scene. The results can be applied for designing medical AR training and educational applications. Fig. 8 shows an application of the developed method. The medical AR scene

is presented to the observer using an "AR window" [20].

Samset et al. [12] present tools based on novel concepts in visualization, robotics and haptics providing tailored solutions for a range of clinical applications. Examples from radio-frequency ablation of liver-tumours, laparoscopic liver surgery and minimally invasive cardiac surgery will be presented.

Navab et al. [13], [14] introduce the concept of a laparoscopic virtual mirror: a virtual reflection plane within the live laparoscopic video that is able to visualize a reflected side view of the organ and its interior. The Laparoscopic Virtual Mirror is able to reflect virtually the 3D volume as well as the laparoscope or any other modelled and tracked instruments. Combining this visualization paradigm with a registration-free augmentation system for laparoscopic surgery, it is possible to get a powerful medical augmented reality system that could make minimally invasive surgeries easier and safer to perform.

Kalkofen et al. [15] overlay carefully synthetic data on top of the real world imagery by taking into account the information that is about to be occluded by augmentations as well as the visual complexity of the computer-generated augmentations added to the view. They solve the problem of augmentations occluding useful real imagery with edges extracted from the real video stream.

De Paolis et al. [16] present an Augmented Reality system that can guide the surgeon in the operating phase in order to prevent erroneous disruption of some organs during surgical procedures. Since the simple augmentation of the real scene cannot provide information on the depth, a sliding window is provided in order to allow the occlusion of part of the organs and to obtain a more realistic impression that the virtual organs are inside the patient's body. In addition, distance information is provided to the surgeon and an informative box is shown in the screen in order to visualize the distance between the surgical instrument and the organ concerned. When the distance between the surgical instrument and some specified organs is under a safety threshold, a video feedback as well as an audio feedback in the form of an impulse are provided. The frequency of this impulse increases when the distance between the surgical instrument and the organ concerned decreases.

Soler et al. [17] present the results of their research into the application of AR technology in laparoscopic and NOTES (Natural Orifice Transluminal Endoscopic Surgery) procedures. They have developed two kinds of AR software tools (Interactive Augmented Reality and Fully Automatic Augmented Reality) taking into account a predictive deformation of organs and tissues during the breathing cycle of the patient. A preclinical validation has been performed on pigs and results are very encouraging and represent the first phase for surgical gesture automation that will make it possible to reduce surgical mistakes.

The collaboration between the MIT Artificial Intelligence Lab and the Surgical Planning Laboratory of Brigham [18] has led to the development of solutions that support the preoperative surgical planning and the intraoperative surgical guidance.

Papademetris et al. [19] describe the integration of image analysis methods with a commercial image-guided navigation system for neurosurgery (the BrainLAB VectorVision Cranial System).

III. THE INFORMED CONSENT

In the current climate of increasing awareness, patients are demanding more knowledge of the operative process. The term "informed consent" explains the process by which, before treatment, comprehensive and impartial information regarding a planned operative procedure is provided to a patient so that he can understand the implications of the procedure before consenting.

Informed consent is a process of communication between patient and physician that results in the patient's authorization or agreement to undergo a specific medical intervention.

In the communications process the physician discusses with the patient about the patient's diagnosis (if known), the nature and purpose of a proposed treatment or procedure, the risks and benefits of a proposed treatment or procedure, the risks and benefits of an alternative treatment or procedure and the risks and benefits of not receiving or undergoing a treatment or procedure.

Bollschweiler et al. [20] present the results of the study of a new method of consenting improved using a multimedia-based information program (MM-IP). 80 patients undergoing laparoscopic cholecystectomy went through the standard informed consent process and a group of patients were also given access to a MM-IP. Questionnaires were used to evaluate the effectiveness of the MM-IP for improving the consent process and were completed before surgery in order to evaluate how patients perceived their own understanding of important aspects of their illness. Patients positively evaluated the use of the MM-IP.

Eggers et al. [21] present a multimedia program aimed at obtaining informed consent from obese patients before gastric banding. The result emphasizes that the multimedia program clearly benefits both surgeons and patients, but the personal contact with the surgeon remains essential because the information presented in multimedia format do not alleviate patient anxiety.

Wilhelm et al. [22] evaluate the impact of an extended education on patients undergoing cholecystectomy. For extended patient information, a professionally built DVD was used and the quality of education was evaluated using a purpose-built questionnaire. They prove the positive impact of an information DVD on patients knowledge;

nevertheless, they assert that the multimedia tools cannot replace personal interaction and should only be used to support daily work.

IV. THE 3D MODELS OF PATIENT'S ORGANS

In MIS, the use of images registered to the patient is a prerequisite for both the planning and the guidance of this kind of operations. From the medical image of the patient (MRI or CT) it is possible to obtain an efficient 3D reconstruction of his anatomy and improve the standard slice view by means of the visualization of the 3D models of the organs.

The 3D models of the patient's anatomy are built from the medical images (MRI or CT) of a patient by means of the application of segmentation and classification algorithms. The grey levels in the medical images are replaced by colours and associated to the different organs.

Several research teams deal with the task of segmentation and developed techniques that allow extracting the patient's organs from CT-scan or MRI automatically or interactively.

Nowadays there are different software used in medicine for the visualization and the analysis of scientific images and the 3D modelling of human organs; Mimics [23], 3D Slicer [24], ParaView [25] OsiriX [26] and ITK-SNAP [27] play an important role among these tools.

In our case studies, the 3D models of the patient's organs have been reconstructed using segmentation and classification algorithms provided by ITK-SNAP and by 3D Slicer.

ITK-SNAP provides semi-automatic segmentation using active contour methods as well as manual delineation and image navigation; it also fills a specific set of biomedical research needs.

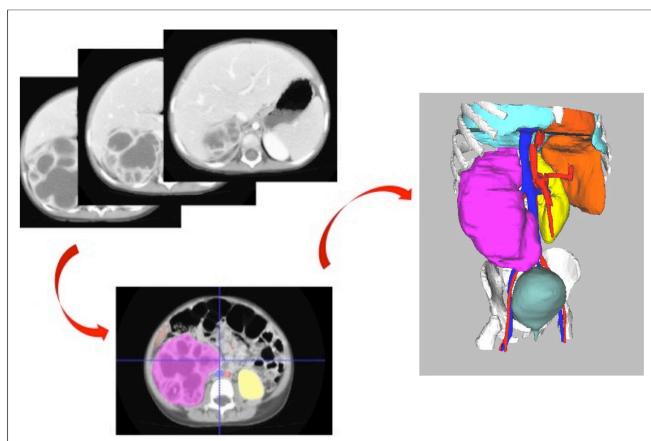


Figure 2. An example of the image processing using ITK-SNAP.

3D Slicer is a multi-platform, free and open-source software package for visualization and medical image computing. The platform provides functionality for segmentation, registration and three-dimensional

visualization of multi-modal image data.

Fig. 2 shows the result of the image processing using ITK-SNAP; the skin and the muscles of the abdominal region are displayed in total transparency and the tumour is shown in magenta.

V. THE CASE STUDIES

We processed two different case studies: the first case study, shown in Fig. 3, concerns a two-year-old child with a benign tumour of the right kidney; the second case study, shown in Fig. 4, concerns a twelve-year-old child with a tumour of the peripheral nervous system (ganglioneuroma).

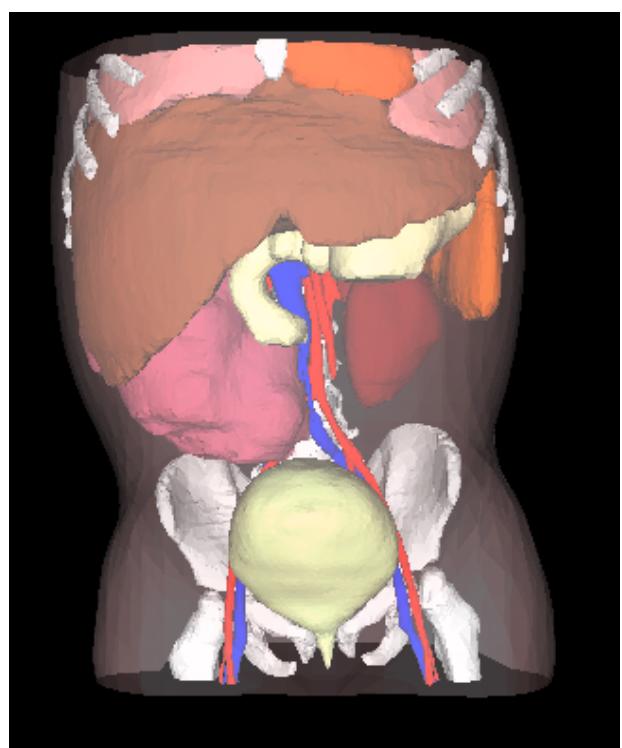


Figure 3. 3D model of a child with a tumour at the kidney.

The slice thickness equal to 3 mm has caused some aliasing effects on the reconstructed 3D models that could lead to inaccuracies. Therefore we have paid special attention to the smoothing of the reconstructed models in order to maintain a good correspondence with the real organs.

In our application, the mesh editing has been carried out using the open-source MeshLab software application [28].

A radiologist has validated the obtained 3D models.

VI. THE USED TECHNOLOGIES

In the application, it is necessary to use an optical tracker in order to detect without delay the right position and orientation of the surgical tool used by the surgeon. The tracking system is also used in order to permit the overlapping of the virtual organs on the real ones in the

augmented visualization of the scene during the real surgical procedure.

Among the different tracking systems based on mechanical, optical or visual technologies, we chose an optical tracker (the Polaris Vicra of the NDI Inc.) in order to avoid the problems typical of the mechanical systems associated to the use of metal devices.

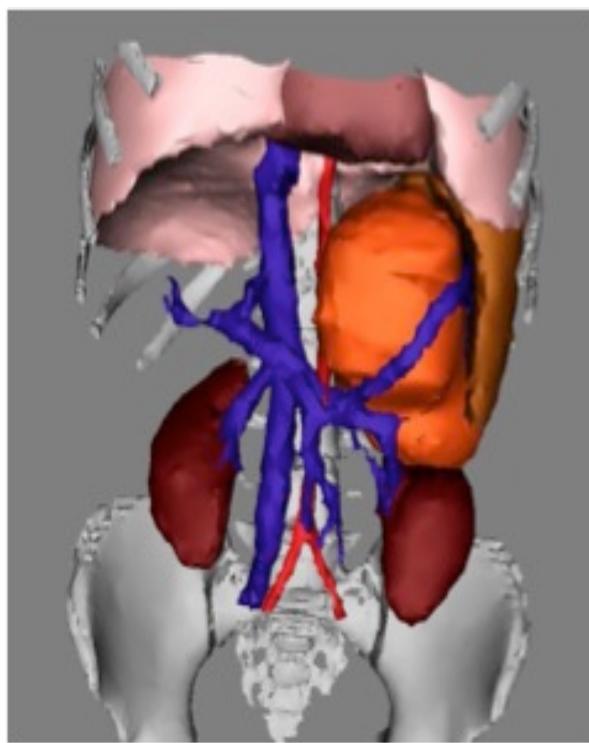


Figure 4. Virtual model of a child with a ganglioneuroma.

The Polaris Vicra optical system [29] tracks both active and passive markers and provides precise, real-time spatial measurements of the location and orientation of an object or tool within a defined coordinate system.

The system uses a position sensor to detect infrared-emitting or retro-reflective markers affixed to a tool or object; using the information received from the markers, the sensor is able to know position and orientation of the tools within a specific measurement volume. The system consists of 2 IR cameras and some tools with reflective beads placed on known geometry frames. The system can calculate the real position of the tool in the space with an accuracy of 0.2 mm and 0.1 of a degree.

The tracking technology is usually used in the modern operating rooms and provides an important help to enhance the performance during the real surgical procedures.

VII. THE USER INTERFACE

The developed application is supplied with a specific user interface that allows the user to take advantage of the feature offered by the software. The application is provided

of 4 sections with the aim to provide support to the surgeons in the different steps of the surgical procedure such as the study of the case, the diagnosis, the pre-operative planning, the choice of the trocar entry points and the simulation of the surgical instruments interaction.

Starting from the models of the patient's organs, the surgeon can note some data about the patient, collect information about the pathology and the diagnosis, choose the most appropriate positions for the trocar insertion and overlap these points on the patient's body using the Augmented Reality technology.

By means of the user interface it is possible to display all the organs of the abdominal region or just some of these using the show/hide functionality; it is also possible to change the transparency of each organ.

It is possible to use this platform in order to describe the pathology, the surgical procedure and the consequent risks to the child's parents, with the aim of obtaining informed consent for the surgical procedure.

VIII. THE DEVELOPED APPLICATION

In the developed application, as shown in Fig. 5, all the patient's information (personal details, diseases, specific pathologies, diagnosis, medical images, 3D models of the organs, notes of the surgeon, etc.) are structured in a XML file associated to each patient.

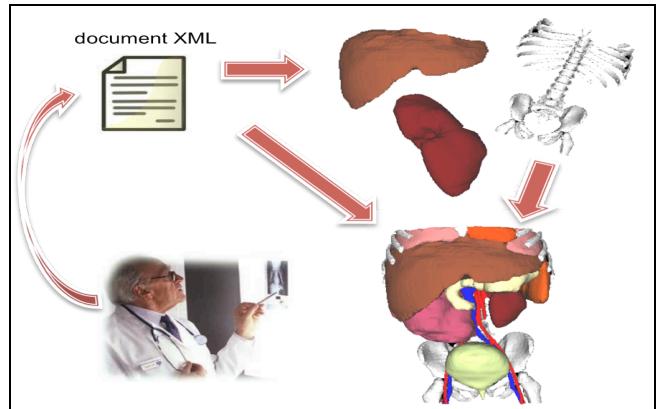


Figure 5. Patient's data collected in an XML file.

A specific section for the pre-operative planning includes the visualization of the virtual organs. The physician can get some measurements of organ or pathology sizes and some distances.

For the computation of the distance between a pair of points we have used the PQP library (Proximity Query Package) [30]. This section is shown in Fig. 6.

By means of a detailed view of the 3D model, the surgeon can choose the trocar entry points and check if, with this choice, the organs involved in the surgical procedure can be reached and the procedure can be carried out in the best way.

Fig. 7 shows the specific section of the user interface for the interaction with the 3D models of the patient's organs.

By means of a detailed view of the 3D model, the surgeon can choose the trocar entry points and check if, with this choice, the organs involved in the surgical procedure can be reached and if the choice allows carrying out the procedure in the best way [31].

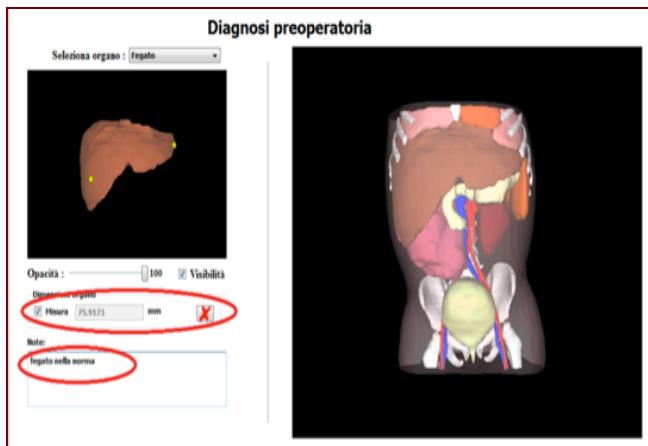


Figure 6. Example of a measurement of organs.

Complications associated with starting first abdominal entry are the first concern for laparoscopic surgeons. In order to minimize first access-related complications in laparoscopy, several techniques and technologies have been introduced in the last years.

The problem of blind access is that it may imply vascular injuries caused by the blind entry of instruments in the abdominal cavity. This problem can be solved with the direct visualization of under-layer viscera and vessels.

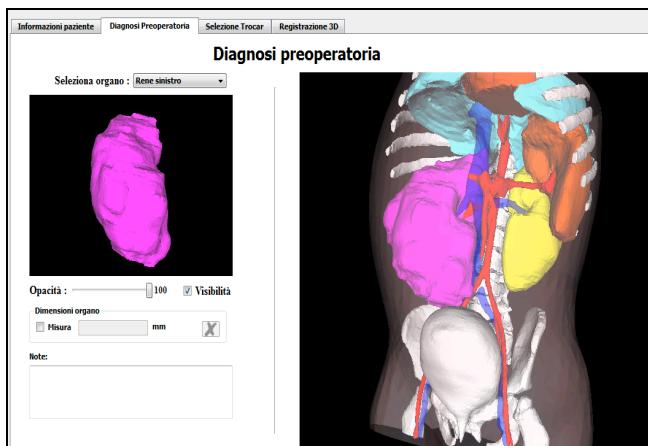


Figure 7. Section for the interaction with the organs.

Sometimes, using the standard insertion points for the surgical tools, also a simple surgical procedure can be very difficult because of the specific anatomy of the different

patients. The surgeon can find it difficult to reach the specific organ or to interact with the surgical tools. In this case he has to choose another insertion point in order to be able to carry out the surgical procedure in the most suitable way.

Our aim is to avoid the occurrence of this situation during the real surgical procedure using the visual information provided by means of the 3D models of the patient's anatomy.

In the developed application, in order to verify if the chosen insertion points allow reaching properly the specific organ interested to the surgical operation and permitting to carry out the procedure in a correct way, it is also possible to simulate the interaction of the surgical instruments.



Figure 8. Section for the choice of the trocar insertion.

Our application, by means of an Augmented Reality module, supports the placement of the trocars on the real patient during the surgery procedure and simulates the insertion of the trocars in the patient body in order to verify the correctness of the chosen insertion sites.

The Augmented Reality surgery guidance aims to combine a real view of the patient on the operating table with virtual renderings of structures that are not visible to the surgeon. In this application we use the AR technology in order to visualize on the patient's body the precise location of selected points on the virtual model of the patient.

For the augmented visualization, in order to have a correct and accurate overlapping of the virtual organs on the real ones, a registration phase is carried out; this phase is based on fiducial points and on the use of an optical tracker.

Fig. 8 shows the section for the accurate choice of the trocar insertion points.

Using the augmented visualization, the chosen entry points for the trocars can be visualized on the patient's body through the Augmented Reality technique in order to support the physician in the real trocar insertion phase.

Fig. 9 shows the specific section for the simulation of the

surgical tools interaction with the possibility to move the trocar entry points using the arrows.

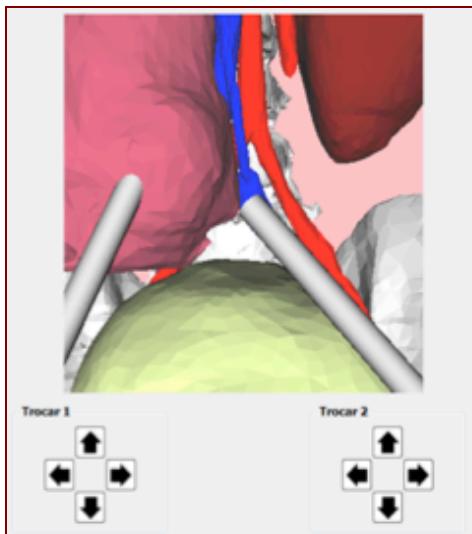


Figure 9. Simulation of the surgical tools interaction.

Fig. 10 shows the augmented visualization of the chosen trocar entry points overlapped on the patient's body (a dummy). The yellow points are the fiducials used for the registration phase and the red ones are the trocar insertion points.

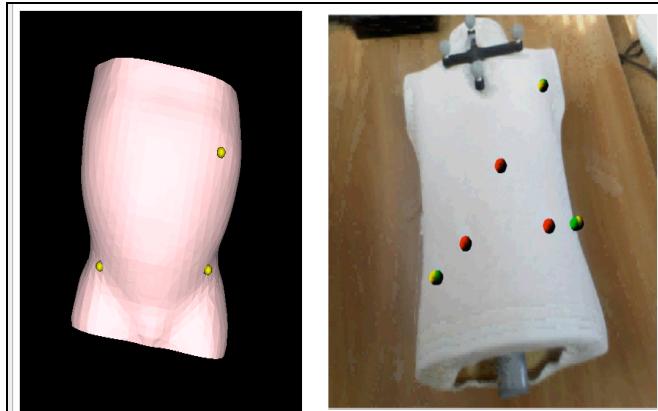


Figure 10. The augmented visualization.

IX. USABILITY TESTS

In order to evaluate the validity and the usability of the developed application and to receive possible suggestions from the users, some tests have been carried out. The test phase has been realized in order to allow the users to check all the functionalities of the application.

After a short period of training (5 minutes), the users have tried to carry out different procedures and, subsequently, they have reported the impressions on a specific questionnaire. 15 subjects have been testing the

application for an average time of 7 minutes and 43 seconds.

The obtained results can be considered satisfactory and some annotations to improve the user interface and the usability of the application have been considered. In particular, the users have suggested:

- To improve the session for the choice of trocar entry points by means of a more accurate explication about the use of the arrows in the interface;
- To provide a more simple way to store the measurements of the organs.

Fig. 11 shows a graph with the test answers about the usability of the different sessions of the application.

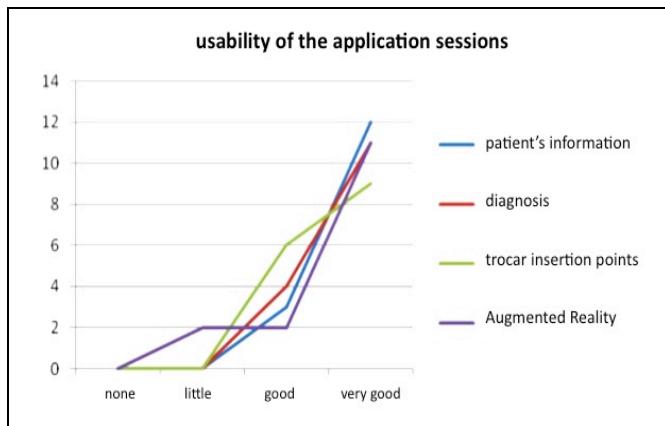


Figure 11. Test answers about the usability of the application sessions.

X. CONCLUSIONS AND FUTURE WORK

The developed platform offers a tool to visualize the 3D reconstructions of the patient's organs, obtained by the segmentation of a CT scan.

The system allows interacting with the models in order to choose the more appropriate insertion points of the trocars and to simulate the placement of these in order to verify the validation of this choice. The Augmented Reality module supports the placement of the trocars on the real patient's body during the surgery procedure.

An accurate integration of the virtual organs in the real scene is obtained by means of an appropriate registration phase based on fiducial points fixed onto the patient. In addition, a complete user interface allows a simple and efficient utilization of the developed application.

Furthermore the platform permits to store the patient and the pathology information that the surgeon can note during the use.

The platform can support the physician in the diagnosis step and in the preoperative planning when a laparoscopic approach will be followed. In addition, this support could lead to a better communication between physicians and patient's parents in order to obtain their informed consent.

The building of a complete Augmented Reality system

that could help the surgeon during the other phases of the surgical procedure has been planned as future work; the acquisition in real time of a patient's video and the dynamically overlapping of the virtual organs to the real patient's body will be developed taking into account the surgeon point of view and the location of medical instrument.

An accurate AR visualization modality will be developed in order to provide a realistic depth sensation of the virtual organs in the real body.

Accuracy and usability tests will be also carried out.

ACKNOWLEDGEMENT

This work is part of the ARPED Project (Augmented Reality Application in Pediatric Minimally Invasive Surgery) funded by the Fondazione Cassa di Risparmio di Puglia. The aim of the ARPED Project is the design and the development of an Augmented Reality system that can support the pediatric surgeon through the visualization of anatomical structures of interest during the pre-operative planning and the laparoscopic surgical procedure.

REFERENCES

- [1] L. T. De Paolis, M. Pulimeni, and G. Aloisio, "An Augmented Reality Application for the Enhancement of Surgical Decisions", The 4th International Conference on Advances in Computer-Human Interactions (ACHI 2011), February 23-28, 2011, Gosier, Guadeloupe, France, pp. 192-196.
- [2] L. T. De Paolis and G. Aloisio, "Augmented Reality in Minimally Invasive Surgery", Advances in Biomedical Sensing, Measurements, Instrumentation and Systems, Lecture Notes in Electrical Engineering, Vol. 55, Mukhopadhyay S.C. & Lay-Ekuakille A. (Eds.), Springer Publisher, December 2009, ISBN 978-3-642-05166-1;
- [3] R. Azuma, "A Survey of Augmented Reality. Presence: Tele-operators and Virtual Environments", 4(6), pp. 355-385, 1997.
- [4] P. Milgram and F. Kishino, "A Taxonomy of Mixed Reality Visual Displays", IEICE Transactions on Information Systems, E77-D(12), 1994, pp. 1321-1329.
- [5] J. B. A. Maintz and M. A. Viergever, "A survey of medical image registration", Medical Image Analysis, vol. 2, 1998, pp. 1-36.
- [6] F. Sauer, "Image Registration: Enabling Technology for Image Guided Surgery and Therapy", 2005 IEEE Engineering in Medicine and Biology, Shanghai, China, 2005.
- [7] M. Feuerstein, S. M. Wildhirt, R. Bauernschmitt, and N. Navab, "Automatic Patient Registration for Port Placement in Minimally Invasive Endoscopic Surgery", Medical Image Computing and Computer-Assisted Intervention (MICCAI 2005). Lecture Notes in Computer Science 3750, Springer-Verlag, Palm Springs, CA, USA, 2005, pp. 287-294.
- [8] T. Sielhorst, M. Feuerstein, and N. Navab, "Advanced Medical Displays: A Literature Review of Augmented Reality", IEEE/OSA Journal of Display Technology, Special Issue on Medical Displays, 4(4), 2008, pp. 451-467.
- [9] F. Devernay, F. Mourguès, and E. Coste-Manière, "Towards Endoscopic Augmented Reality for Robotically Assisted Minimally Invasive Cardiac Surgery", IEEE International Workshop on Medical Imaging and Augmented Reality, 2006, pp. 16-20.
- [10] C. Bichlmeier and N. Navab, "Virtual Window for Improved Depth Perception in Medical AR", International Workshop on Augmented Reality environments for Medical Imaging and Computer-aided Surgery (AMI-ARCS), Copenhagen, Denmark, 2006.
- [11] C. Bichlmeier, F. Wimmer, H. S. Michael, and N. Nassir, "Contextual Anatomic Mimesis: Hybrid In-Situ Visualization Method for Improving Multi-Sensory Depth Perception in Medical Augmented Reality", Sixth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR '07), 2007, pp. 129-138.
- [12] E. Samset, D. Schmalstieg, J. Vander Sloten, A. Freudenthal, J. Declerck, S. Casciaro, Ø. Rideng, and B. Gersak, "Augmented Reality in Surgical Procedures", SPIE Human Vision and Electronic Imaging XIII, (6806):68060K.1-68060K.12, 2008.
- [13] N. Navab, M. Feuerstein, and C. Bichlmeier, "Laparoscopic Virtual Mirror - New Interaction Paradigm for Monitor Based Augmented Reality", IEEE Virtual Reality Conference 2007 (VR 2007), Charlotte, North Carolina, USA, 2007, pp. 10-14.
- [14] C. Bichlmeier, S. M. Heining, M. Rustae, and N. Navab, "Laparoscopic Virtual Mirror for Understanding Vessel Structure: Evaluation Study by Twelve Surgeons", 6th IEEE International Symposium on Mixed and Augmented Reality (ISMAR'07), Nara, Japan, 2007.
- [15] D. Kalkofen, E. Mendez, and D. Schmalstieg, "Interactive Focus and Context Visualization in Augmented Reality", 6th IEEE International Symposium on Mixed and Augmented Reality (ISMAR'07), Nara, Japan, 2007, pp. 191-200.
- [16] L. T. De Paolis, M. Pulimeni, M. Lapresa, A. Perrone, and G. Aloisio, "Advanced Visualization System Based on Distance Measurement for an Accurate Laparoscopy Surgery", Joint Virtual Reality Conference of EGVE - ICAT - EuroVR, Lyon, France, 2009.
- [17] L. Soler, S. Nicolau, J.-B. Fasquel, V. Agnus, A. Charnoz, A. Hostettler, J. Moreau, C. Forest, D. Mutter, and J. Marescaux, "Virtual Reality and Augmented Reality Applied to Laparoscopic and NOTES Procedures", IEEE 5th International Symposium on Biomedical Imaging: from Nano to Macro, 2008, pp. 1399-1402.
- [18] W. E. L. Grimson, T. Lozano-Perez, W. M. Wells, G. J. Ettinger, S. J. White, and R. Kikinis, "An Automatic Registration Method for Frameless Stereotaxy, Image Guided Surgery, and Enhanced Reality Visualization", Transactions on Medical Imaging, 1996.
- [19] X. Papademetris, K. P. Vives, M. Di Stasio, L. H. Staib, M. Neff, S. Flossman, N. Frielinghaus, H. Zaveri, E. J. Novotny, H. Blumenfeld, R. T. Constable, H. P. Hetherington, R. B. Duckrow, S. S. Spencer, D. D. Spencer, J. and S. Duncan, "Development of a research interface for image guided intervention: Initial application to epilepsy neurosurgery", International Symposium on Biomedical Imaging ISBI, 2006, pp. 490-493.
- [20] E. Bollschweiler, J. Apitzsch, R. Obliers, A. Koerfer, S. P. Mönig, R. Metzger, and A. H. Hölscher, "Improving informed consent of surgical patients using a multimedia-based program? Results of a prospective randomized multicenter study of patients before cholecystectomy", Annals of Surgery, August 2008, 248(2), pp. 205-11.
- [21] C. Eggers, R. Obliers, A. Koerfer, W. Thomas, K. Koehle, A. H. Hölscher, and E. Bollschweiler, "A multimedia tool for the informed consent of patients prior to gastric banding obesity", Silver Spring, November 2007, 15(11), pp. 2866-2873.
- [22] D. Wilhelm, S. Gillen, H. Wirnhier, M. Kranzfelder, A. Schneider, A. Schmidt, H. Friess, and H. Feussner, "Extended

- preoperative patient education using a multimedia DVD-impact on patients receiving a laparoscopic cholecystectomy: a randomised controlled trial”, Langenbeck’s Archives of Surgery, March 2009, 394(2), pp. 227-33.
- [23] Mimics Medical Imaging Software, Materialise Group. Available: <http://www.materialise.com/mimics>
- [24] 3D Slicer. Available: <http://www.slicer.org>
- [25] J. Ahrens, B. Geveci, and C. Law, “ParaView: an End-User Tool for Large Data Visualization”, Visualization Handbook, Edited by C.D. Hansen and C.R. Johnson, Elsevier, 2005.
- [26] O. Fahy, “Osirix: an Open Source Platform for Advanced Multimodality Medical Imaging”, 4th International Conference on Information & Communications Technology, Cairo, Egypt, 2006, pp. 1-2.
- [27] P. A. Yushkevich, J. Piven, H. Cody, S. Ho, J. C. Gee, and G. Gerig, “User-Guided Level Set Segmentation of Anatomical Structures with ITK-SNAP”, Insight Journal, Special Issue on ISC/NA-MIC/MICCAI Workshop on Open-Source Software, Nov 2005.
- [28] P. Cignoni, M. Callieri, M. Corsini, M. Dellepiane, F. Ganovelli, and G. Ranzuglia, “MeshLab: an Open-Source Mesh Processing Tool,” in Proc. Sixth Eurographics Italian Chapter Conference, 2008, Salerno, Italy, pp. 129-136.
- [29] NDI Polaris Vicra. Available: <http://www.ndigital.com>
- [30] E. Larsen, S. Gottschalk, C. L. Ming, and D. Manocha, “Fast Proximity Queries with Swept Sphere Volumes”, Technical Report TR99-018, Dept. of Computer Science, University of North Carolina, 1999.
- [31] A. Tinelli, A. Malvasi, G. Hudelist, O. Istre, and J. Keckstein, “Abdominal Access in Gynaecological Laparoscopy: a Comparison between Direct Optical and Open Access” Journal of Laparoendosc & Advanced Surgical Techniques, 19(4), 2009, pp. 529-33.

Retrieval of 3D Medical Images via Their Texture Features

Xiaohong Gao, Yu Qian, Martin Loomes, Richard Comley, Balbir Barn,
Alex Chapman, Janet Rix
Middlesex University, London, NW4 4BT, UK
{x.gao, y.qian, m.loomes, r.comley, b.barn, a.chapman, j.rix}@mdx.ac.uk

Rui Hui, Zengmin Tian
General Navy Hospital, Beijing, P.R. China
huirui2002@163.com, tianzengmin@vip.sina.com

Abstract -- While content-based image retrieval has been researched for more than two decades, retrieving 3D datasets has been progressing considerably slower, especially in the application to the medical domain. This is in part due to the limitation of processing speed while trying to retrieve high-resolution datasets in real-time. Another barrier is that most existing methods have been developed based on 2D images instead of 3D, leaving a gap to be filled. At present, a significant number of exploitations are focusing on the extraction of 3D shapes. As it happens, it appears that, to a large extent, the remaining information tends to be equally important in the task of clinical decision making. With this in mind, in this paper, a texture-based online system, MIRAGE, has been developed to facilitate CBIR for 3D images. Specifically, four texture-based approaches stemming from 2D forms are studied extensively through the application to 3D images using a collection of MR brain images and are implemented, which include 3D Local Binary Pattern (LBP), 3D Grey Level Co-occurrence Matrices (GLCM), 3D Wavelet Transforms (WT) and 3D Gabor Transforms (GT). Based on the nature of the content, each approach has its own advantages and disadvantages. For example, in terms of retrieval precision of tumours and processing speed, LBP not only achieves precision rate of up to 78% but also can perform retrieval in real time with sub-second processing speeds, outperforming the others.

Keywords – CBIR; 3D image retrieval; 3D texture extraction; MIRAGE system; 3D visualization.

I. INTRODUCTION

Due to the advances of medical imaging techniques, more and more images are in three (or higher) dimensional forms, allowing a coherent and collective view. Since many of these images are comprised of 2D slices, most current databases archive and index them in 2D form, especially for the systems that are indexed by their content. As a result, a number of limitations have arisen with the most significant one being that the information extracted from a single 2D slice cannot be representative due to the fact that slices are getting thinner and thereafter resolutions are getting higher.

On the other hand, at present, content-based retrieval for three dimensional (3D) images has been researched primarily to meet the demand for 3D pictures available over the internet. In this way, the main challenge facing the extraction of features from 3D images is that these features have to be invariant of viewing angles, i.e., invariant of rotation, in order to achieve higher retrieval hit of similar objects, even though sometimes

they may not be visible from all the viewing angles. For example, if a query image is a 3D rabbit with a head facing the view, a good retrieval system should bring back relevant objects including those showing only its tails as an exact match. In other words, even if the view angle is at the back of the object, the matched objects can still be found. In addition, in 2D cases, the viewing angle is always at 0°, being normal to the computer screen, by which most existing algorithms can fulfill this request. Also, many of the other characteristics of content-based image retrieval (CBIR) are shared between 2D and 3D, including scaling and translation of regions of interest. This has led to the shift of many current studies to focusing on the invariance of transformations (including rotation, scaling and translation) of objects, which has more to do with shapes.

A. 3D CBIR for Non-medical Images

Since the emerging of the internet in 1990, coupled with the advance of computer hardware, vast amounts of textual and imagery data are available online, prompting the creation of an array of text-based search engines, such as Google and Yahoo, in an attempt to filter the relevant data. For image data, however, thanks to their embedded information being inside the pictures, a text-based approach has its limitations, especially for those images that are not properly, if not at all, labelled. Consequently, CBIR has been researched both horizontally and vertically. As the trend continues, the progress in the last century (1994-2000) has been very well documented in [1], whereas the state of the art in the last decade (2001-2008) was reviewed by [2] with a number of future directions being identified. Generally, a CBIR system follows the procedures of development as shown next. Firstly, it extracts features of images in terms of their global visual information, such as colour, texture, and shape. Then, these features are represented using mathematical vectors that, in turn, are employed to index each image. Finally, when a query image is submitted, the system needs to extract these features from the query image and to perform the comparison with the feature database that has been stored in advance. In this way, the retrieval process of an image can be as fast as that in a text-based system since the similarity calculation is based on numerical data.

For 3D online images, the majority of approaches concerns with the features of shapes as an indexing key. For example, in [3], 3D Zernike descriptors have been developed to describe

shapes of objects, by taking advantages of polynomial representations, on which these descriptors are based, being invariant of transformations. To this end, a database has to constitute objects differentiated by shapes, such as airplanes, chairs, etc.. Similarly, in order to achieve transformation invariance, a graph-based shape descriptor is created in [4] in an effort to determine the way to calculate a similarity between 3D objects. Recently, the retrieval of 3D objects has been attempted using impact descriptors [5] attempting to capture the surrounding areas of a 3D shape in order to offer a histogram of time-space curvature, which are invariant of rotation and translation. Elsewhere, other shape-based 3D models are included in [6-9]. Because shape-based approaches only describe the surface of a 3D object, they tend to ignore the content inside that object. Depth based descriptors therefore have been developed as demonstrated in [10], which is however in principle, still capture the outline of a shape at each depth (z-buffer).

More recently, the approach of scale invariant feature transformation, commonly known as SIFT, has attracted substantial attention. Originally developed for 2D images [11], SIFT has been extended to 3D spaces in an attempt to perform action recognition [12] in a video sequence and object recognition for an airport security checking [13].

B. 3D CBIR for Medical Images

Progress on CBIR for 3D images have been reviewed by many researchers [14] with several developed systems demonstrated, which are summarized in Table 1 and described in details below.

TABLE 1: 3D CBIR SYSTEMS OF MEDICAL IMAGES

Name/ Feature	Imaging Modality	Domain	Reference
QBISM / intensity-based	MRI/PET	Brain	Arya [15]
Pre-defined-semantic-based	CT	Brain	Liu [16]
MIMS / ontology-based	All	All	Chbeir [17]
Knowledge-based	All	All	Chu [18]
ILive – modality-based	All	All organs	Mojisilovic [19]
2D Texture-based	MR	Heart	Glatard [20]
FICBDS / Physiological information based	Functional PET	Brain	Cai [21]
3D PET / lesion-based	PET	Brain	Batty [22]
MIRAGE / 3D texture-based	MR	Brain	Gao [23], Qian[24]

In the system of QBISM, 3D functional brain images are queried and visualized [15], by which intensity-based volume data are stored for spatial references, whereas Talairach brain atlas [25] is employed to construct a region-based retrieval. The key to this system is the application of volumetric data type, i.e., the Region or Volume being expressed as $\langle x, y, z, \text{value} \rangle$, in

the representation of image data, which in some cases, might be prone to noise.

In other cases, the retrieval task of 3D images can work well based on feature extraction [16] from 2D slices, whose success to a great extent, is dependent on the application fields of the created databases.

On the other hand, semantics based retrieval remains acceptable to images of all dimensions as evidenced by [19]. The strength of this work therefore lies in the approaches employed for categorization of images that bear semantically well-defined data sets. This task itself however in most cases poses greater challenges than semantic representation itself. Nevertheless, semantics based retrieval of medical images offers one of the current trends. Likewise, ontology-based [17] and knowledge-based approaches [18] can shorten the semantic gap to a certain extent between low level features and high level semantics, which in turn requires skilful expertise, i.e., in-depth knowledge, to interpret images and convert contents into textual descriptions.

For subject-based images that bear centralised characteristics, local features can play an important part in indexing and retrieving images. For example, the system of FICNDS [21] employs physiological kinetic features for retrieving images. Similarly, Batty and Gao [22] have employed binding potential (BP) values to index functional PET images. Although effective, this method is very discipline-defined and relies heavily on the additional supply of extra information. For example, in FICNDS, to define a tracer kinetic model, plasma time activity (PTA) curves should be obtained from a series of blood samples, which are not easily available for most of the images in a database. Although PTA can still be modeled by the application of control regions as applied by Gao et al, a sequence of images acquired over a period of time, say 90 minutes, are still needed, which again is not readily available in most of image repositories. Additionally, in essence, the establishment of kinetic models stems from the data of 2D slices, which may lose information in between slices. For a system that warehousing images of variety of domains, more general approach appears to be in demand in order to be sustainable.

A texture-based approach for retrieving of 3D+ cardiac images has been applied by Glatard [20] with the employment of a 2D Gabor filter. While working on 4D (3D + time) heart images, their adoption of a Gabor filter is again, in essence, a 2D form based on regions coupled with an extra parameter dedicated to myocardium features.

For application to medical images, a Volume of Interest (VOI) consists of not only boundary shapes, but also inside textures representing tissue properties of the VOI. The information extracted from these textures equally plays an important role in describing the VOI and is important to medical doctors at most of the time. Therefore these texture features should be taken into consideration in the representation of an object as well.

Apparently, it is possible to represent texture in 2D-based form, since a 3D dataset constitutes a stack of 2D slices.

However, using a slice-by-slice 2D approach suffers from the disappointment that some important information inter-laced within the volumetric data is missing. Thus, in terms of a 3D form of texture, this spatial structural information should be extracted from a cube instead of a surface or a square. Towards this end, while working on images of 3D brain, Gao et al [23] and Qian et al [24] have furthered four texture-based approaches into 3D form to the domain of medical image retrieval to extracting texture information that is subsequently utilized for indexing them in their developed system MIRAGE [26].

Specifically, in this study, the approach of Local Binary Pattern (LBP) [27] is addressed first because of its discriminative power and computational simplicity, and applied to a collection of 3D MR brain images for extracting texture information that is subsequently utilized for indexing them. Three other well-known methods in texture representation are also investigated, including Grey Level Co-occurrence Matrices (GLCM), Wavelet Transforms (WT) and Gabor Transforms (GT). The novelty of this work demonstrates the feasibility of 3D texture-based approaches for image retrieval while maintaining real time operation. This is achieved by the introduction of a pre-processing stage of a selection of potential VOIs into query datasets; by which, through the use of statistically analysis of the bilateral symmetry of a brain MR image, a potential VOI of a query can be detected in real time, preceding the extraction of 3D texture features and the calculation of similarities.

The remaining of the paper is hence structured in the following pattern. Section II explains the methods employed in the study, which is followed by Section III that shows the experimental results. The interface design is detailed in Section IV, which is succeeded by Section V providing conclusion and discussion. The last two sections give acknowledgment and references respectively.

II. METHODOLOGY

The development of a repository requires two main phases, which are ingestion and retrieval. In this investigation, at the phase of ingestion of the data, the collected data firstly undergo a pre-processing stage to normalize them into the same resolution before the indexing stage, as shown in the flow chart in Figure 1. As illustrated in the diagram, after spatial normalization of volumetric brain data into a standard template, the data are then divided into 64 non-overlapping equally sized blocks, from which, 3D texture features can be extracted to create a feature database. On the query side, a pre-processing stage is introduced to detect a potential VOI after spatial normalization from a query image. As a result, 3D texture features from a query can only be extracted from these potential sub-blocks of VOIs, which, in the retrieval stage, are compared with the corresponding features in the feature database to obtain retrieval results. Details are elaborated in the following subsections.

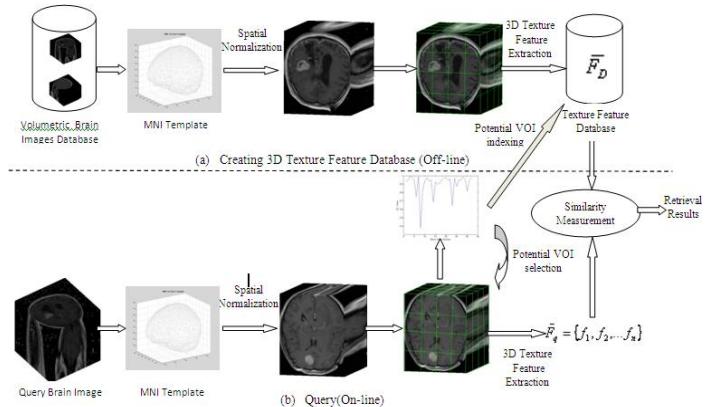


Figure 1. Framework of 3D MR image retrieval.

A. Spatial Normalization

By nature, data are collected from different sources, leading to the fact that brain images vary in both shape and size. In order to make inter-brains comparable, it is necessary to transform the dataset of each individual brain into a standard brain template. In this regard, the software of Statistical Parametric Mapping (SPM5) [28] is employed to spatially normalize a brain image into a template of either MNI T1 or T2 [29] depending on whether an image is acquired by an MR scanner of either T1 or T2 type. In this way, all the images in the database are of the same size with $157 \times 189 \times 69$ voxels.

B. Extraction of Volumetric Textures

In order to describe local features from different parts of a brain, a 3D volumetric brain is divided into 64 non-overlapping equally sized blocks, giving 4 blocks along each of x , y , z axes respectively, as shown in Figure 1. Texture features are then extracted using 3D LBP to create a feature database, upon which image searching and retrieval are performed.

C. 3D Local Binary Pattern

The Local Binary Pattern (LBP) operator is derived from a general definition of texture in a local neighborhood (e.g., 8×8 pixels). In a 2D form, for each pixel in an image, a binary code is produced by thresholding its value with the value of a centre pixel. A histogram is then generated to calculate the occurrences of different binary patterns. To extend this approach to 3D images, similarly to [30], a 3D dynamic texture is recognized by concatenating three histograms obtained from the LBP on three orthogonal planes. When applied to our normalized brain images, they are in the plane of Left-Right (LR), Anterior-Posterior (AP), and Superior-Inferior (SI) respectively, as depicted in Figure 2.

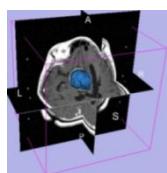


Figure 2. An example of three orthogonal planes in a 3D brain.

These three orthogonal planes intersect in a centre voxel. By selecting 8 neighbours as a local neighbourhood with the radius length being one voxel, fifty-nine uniformed LBP codes are subsequently extracted from the planes of SI, LR and AP respectively, again as illustrated in Figure 2, producing a 59 bin histogram for each plane by accumulating 59 binary patterns. Finally, the three histograms are concatenated to generate a 3D texture representation, giving the size of a feature vector being 177 (=59×3) elements.

D. Lesion Detection

The initial goal of the development of this 3D CBIR system is to search images with lesions of similar location, size or shape (all the collections of images are with lesions). Although a feature database has been implemented in advance, the processing of a query has to be conducted in real time. In other words, after a query is submitted to the system, 3D texture features should be extracted from its 64 sub-volumetric spaces together with the calculation of similarity distances. To this end, while maintaining the overall performance of retrieval, the detection of candidate lesions from sub blocks is carried out first to highlight the abnormalities, such as tumours, with an intention to speed up the retrieval process.

To do this, the characteristics of bilateral symmetry of a brain along its mid-plane (parallel to SI direction as shown in Figure 2) remain assumed. Similarly to [31], by comparing the left half with the right counterpart of a hemisphere along this middle symmetry plane, the abnormality can be envisaged to be singled out. Since a normalized brain image has been divided into 64 blocks, statistical features (e.g., mean, standard deviation, etc.) of each sub-block together with its mirror block are then calculated and compared to establish potentially abnormal sub-blocks.

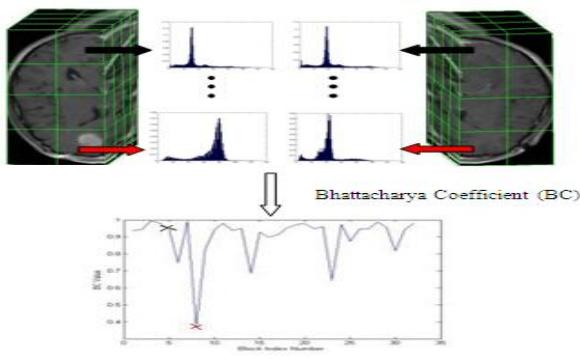


Figure 3. Potential VOI selection

As demonstrated in Figure 3, a normalized brain is divided into left (L) and right (R) parts by a sagittal plane, leading to 32 sub-blocks each, within which a grey level histogram is obtained. The Bhattacharya Coefficient (BC) [32] is thereafter computed between two normalized histograms H_L and H_R , which are obtained from two mirror symmetric sub-blocks as defined in Eq. (1).

$$BC(H_L, H_R) = \sum_i \sqrt{H_L(i) * H_R(i)} \quad (1)$$

The more similar H_L and H_R are, the closer to 1 the BC value is. On the other hand, less similar histograms tend to have smaller BC values. In total, 32 BC values are calculated from 32 paired mirrored symmetric sub-blocks that are plotted at the bottom of Figure 3. The horizontal axis points to the index numbers of sub-block pairs, whereas the vertical axis represents the corresponding BC values. Also shown in the figure are the BC values presenting the top normal sub-block pair marked with a black 'x', whereas the bottom abnormal sub-block pair marked with a red cross. Therefore, the mean value of the BC range works as a threshold to be applied to detect the potentially abnormal sub-block, i.e., where $BC < Threshold$.

After the affirmation of a lesioned VOI from a query is established, 3D texture features are extracted exclusively from this VOI of the query, and are later compared with the features from similar blocks of those images in the feature database in an attempt to search images with similar lesions in terms of textures.

E. Similarity Measurement

To measure the degree of similarity between two images Q and I , a distance function should usually be in place calculating the distance between features of the two images. For a 3D LBP, the histogram intersection is applied to measure features of histograms and is given in Eq. (2),

$$D(Q, I) = \sum_i \min(Q_i, I_i) \quad (2)$$

where i represents each bin in a histogram. The more similar they are between a query (Q) and an image (I), the bigger the value of the D is. Therefore, the retrieved results are ranked in descending order based on the value of D .

III. EXPERIMENTAL RESULTS

A. Data Collection

In this study, the database contains over 100 3D MR brain images with lesions (e.g., tumour, biopsy) and detailed diagnosis. Each dataset has a resolution in a range between $256 \times 256 \times 22 \text{ mm}^3$ and $256 \times 256 \times 44 \text{ mm}^3$, and is in DICOM (Digital Imaging and Communications in Medicine) format with 16 bit grey-level resolution.

B. Results on Detection of Lesions

Since the location of a lesion region plays an important part in retrieving relevant datasets, the evaluation on the detection of lesion positions is carried out first. In Table 2, the first row is the labelling number of the location of a VOI assigned by the authors for the convenience of calculations, e.g., '1' refers to the abnormal part in the front top left part of the brain. The second row is the total number of images containing VOIs in the same positions in the database, whilst the number of correctly detected images by the approach of lesion detection as explained in Section II.D is given on the third row. Therefore, the overall performance in terms of VOI locations is calculated as the number of detected positive VOIs divided by the total positive VOIs and is 91.3% (=168/184).

TABLE 2 VOI DETECTION RATE

VOI Location	1	2	3	4	5	6	7	8	Total
Number of images	24	46	18	38	24	12	14	8	184
Correctly detected images	24	42	16	34	24	8	12	8	168
Correct Detection Rate (%)									91.3

C. Comparison with the Other Texture-based Approaches

The other three methods widely employed in texture representations are also exploited in this investigation by the extension to 3D, including Grey Level Co-occurrence Matrices (GLCM), Wavelet Transforms (WT), Gabor Transforms (GT), which are summarized next.

In 3D form, GLCM [33, 34] are defined as three dimensional matrices of a joint probability of occurrence of a pair of grey values separated by a displacement $d = (dx, dy, dz)$.

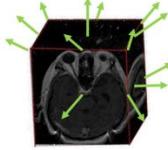


Figure 4.Thirteen directions in 3D GLCM.

For example, four distances with 1, 2, 4, and 8 voxels respectively and thirteen directions, as depicted in Figure 4, which are chosen in this study, will produce 52 (=4*13) displacement vectors, and thereafter 52 co-occurrence matrices. As a result, four Haralick texture features [35], being energy, entropy, contrast and homogeneity, are computed from each matrix, generating a feature vector of 208 components (=4(measures) × 52 (matrices)).

On the other hand, the 3D WT provides a spatial and frequency representation of a volumetric image, which can be achieved by applying both high-pass (H) and low-pass (L) filters along all three dimensions. This is then followed by a 2 to 1 sub-sampling of each output volumetric image [36], giving rise to eight wavelet coefficients sub-bands (one low frequency sub-band and seven high frequency sub-bands) at each scale, as

schematically presented in Figure 5(a). The process is subsequently repeated in the lowest frequency sub-band (LLL_1), generating a 3D wavelet transform of two scales as shown in Figure 5(b).

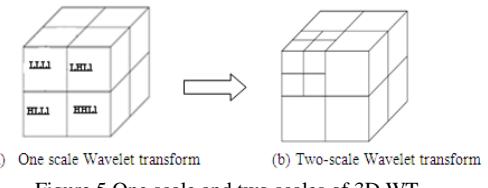


Figure 5.One scale and two scales of 3D WT.

In this investigation, 2 scales of 3D WT, as shown in Figure 5(b) are chosen. The measurement of mean μ and standard deviation σ are then extracted for each sub-band. So that there are 30 features, i.e., 2 (scales) *7 (sub-bands in each scale) *2 (measures) +2 (measures in the lowest resolution) =30, derived from a Wavelet transform of 2 scales, yielding the dimension of a vector being 30.

With respect to Gabor Transforms, in order to extend GT into three dimension, a set of 3D Gabor filters are generated similar to [37, 38] to detect spatial orientations and scale tunable edges and lines (bar), which can be formulated as Eq. (3).

$$g(x, y, z, F, \theta, \phi) = \hat{g}(x, y, z) \exp [j2\pi(F \sin \theta \cos \phi x + F \sin \theta \sin \phi y + F \cos \theta z)] \quad (3)$$

where $\hat{g}(x, y, z)$ is a 3D Gaussian function, together with radial centre frequency F and orientation parameters (θ and ϕ), determining a Gabor filter in three dimensions.

In this study, the following parameters are defined, including four centre frequencies with $F = \{0.0442, 0.0625, 0.0884, 0.125\}$ circle/voxel respectively, six orientation angles, i.e., $\theta = \{0^\circ, 30^\circ, 60^\circ, 90^\circ, 120^\circ, 150^\circ\}$ and six values of ϕ , i.e., $\phi = \{0^\circ, 30^\circ, 60^\circ, 90^\circ, 120^\circ, 150^\circ\}$, which leads to the number of 144 (= 4*6*6) Gabor filters that are employed to extract texture features. Given a 3D volumetric texture $f(x, y, z)$, its 3D Gabor transform GT_i is defined by

$$GT_i = f(x, y, z) * g(x, y, z, F_i, \theta_i, \phi_i) \quad i = 1, 2, 3, \dots, 144 \quad (4)$$

The mean μ and standard deviation σ of GT coefficients are then calculated which act as a representation of texture features from 144 Gabor transforms respectively. Therefore a feature vector includes 288 elements (= 4 (scales) *36 (orientations) *2 (measures)).

To calculate similarity distances from these three methods, a normalized Euclidean distance is employed to compare two 3D patterns in a feature space, as defined by Eq. (5).

$$D(Q, I) = \sqrt{\sum_i \left(\frac{Q_i - I_i}{\sigma_i} \right)^2} \quad (5)$$

where σ_i refers to the standard deviation of a set of representative features over the entire database and is therefore utilized to normalize each individual feature component. The retrieved 3D images are ranked in an ascending order of feature distances.

In summary, the above three 3D texture approaches together with LBP are applied to extract texture features from each sub-volumetric block. Furthermore, the dimension of a feature vector for a 3D brain remains to be the size of local features multiplied by 64, the number of the blocks each volumetric image is divided into, yielding 13312, 1920, 9216 and 11328 components for the approaches of 3D GLCM, 3D WT, 3D GT and 3D LBP respectively.

Subsequently, the performance of image retrieval is evaluated based on the measures of Precision (P) and Recall (R). Precision is defined as the fraction of retrieved images relevant to a query whilst recall is the fraction of relevant images retrieved. Precision and recall values are usually presented together in a Precision-Recall (P-R) graph that demonstrates the retrieval performance at each point in the ranking. In a P-R graph, the horizontal axis refers to a recall whereas the vertical axis shows the corresponding precision at each of the usual recall points, i.e., 10%, 20%, ..., 100% or 0.1, 0.2, ..., 1. A single value, usually, the Mean Average Precision (MAP) value is employed to assess the overall performance for all queries and is calculated as

$$\text{Mean Average Precision(MAP)} = \frac{1}{M} \sum_{i=1}^M AP_i \quad (6)$$

where M is the total number of the queries, AP_i is the average precision for the i^{th} query that is formulated as Eq. (6),

$$\text{Average Precision(AP)} = \frac{1}{N_r} \sum_{j=1}^{N_r} P_j \quad (7)$$

where N_r is the total number of relevant images in a dataset for a query, p_j is the precision when retrieving the j^{th} relevant image.

Figures 6 and 7 depict the average Precision Recall Graph for ten queries across the whole datasets with Figure 6 showing the results without a pre-processing stage of VOI selection whilst Figure 7 with the pre-processing stage.

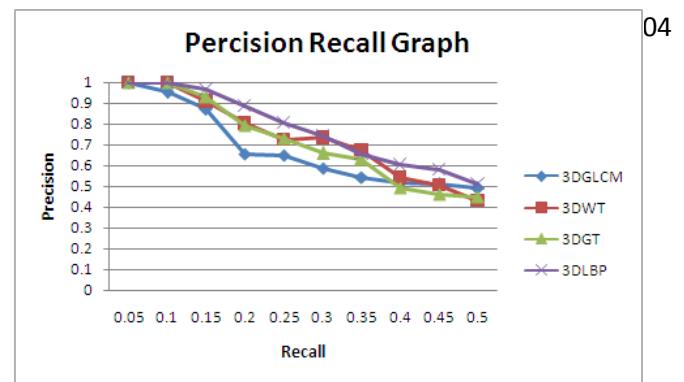


Figure 6. Average precision recall graph for ten queries without VOI selection.

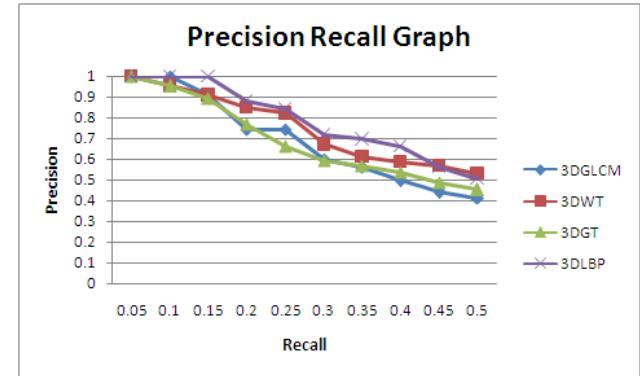


Figure 7. Average precision recall graph for ten queries with VOI selection.

Overall, the mean average precision (MAP) at 0.5 recall rate for ten queries cross the whole database by using the approaches of 3D GLCM, 3D WT, 3D GT and 3D LBP are shown in the following table.

TABLE 3 VALUE OF MEAN AVERAGE PRECISION

Methods	Without VOI selection	With VOI selection
3D GLCM	0.677	0.690
3D WT	0.731	0.749
3D GT	0.714	0.691
3D LBP	0.774	0.786

Comparing the value of MAP with and without potential VOI selection, the methods of 3D GLCM, 3D WT and 3D LBP with potential VOI selection show a slightly improved performance with bigger MAPs.

Figure 8 visualizes the retrieved results by using the four approaches with a pre-processing stage of VOI selection. The query image with a tumour in the middle is displayed in 3D fashion and 3 slices appearing in 3 orthogonal planes on the top row, i.e., in axial, sagittal, and coronal directions. The retrieval results are visualized by using an open source software 3D Slicer [39].

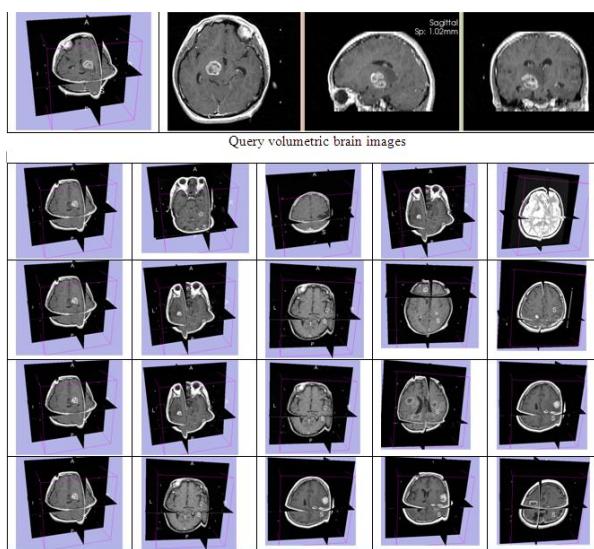


Figure 8. Retrieved results in top 5 ranking from 3D GLCM (row 1), 3D WT (row 2), 3D GT (row 3), and 3D LBP (row 4).

D. Query Time

It is understandable that retrieving images in 3D form might not be performed in real time, one of the drawbacks in the development of CBIR systems for images of higher dimensions. Table 4 demonstrates the average querying time, amounting to the period spent on both feature extraction and retrieval. The second column is the averaged querying time without a pre-processing stage while the third column is with VOI selection, i.e., with a pre-processing stage. All methods are programmed in software of Matlab R2009a running with an Intel P8600 1.58GHz CPU and 3.45GByte RAM.

TABLE 4 QUERY TIME

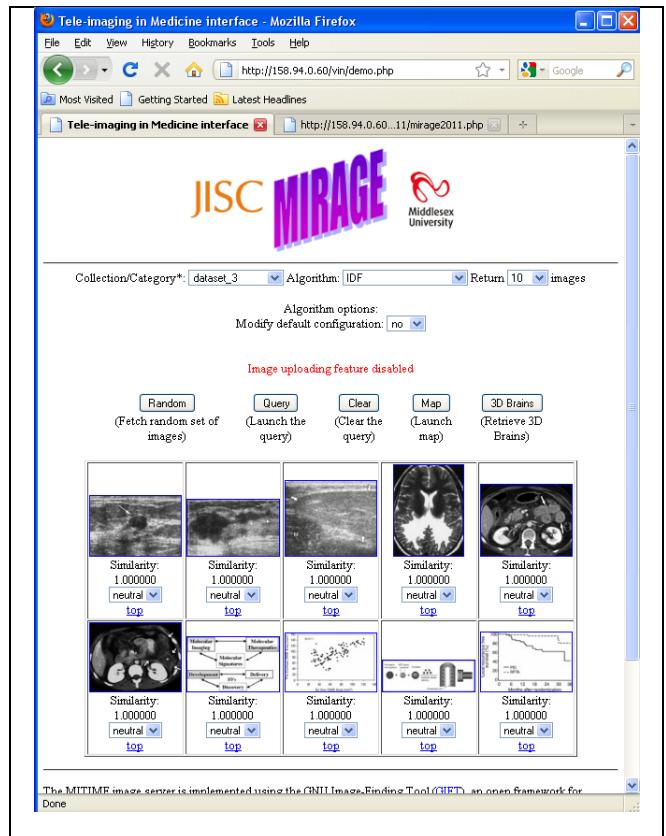
Methods	Without VOI selection	With VOI selection
3D GLCM	43.37s	10.96s
3D WT	4.46s	1.22s
3D GT	38.79m	10.77m
3D LBP	0.74s	0.21s

As can be seen in Table 4, the query time with VOI selection offers 4 times faster operation than that without. In particular, the query time for 3D GT takes much longer than for the other methods spending 38 minutes, due to the employment of 144 times of 3D convolutions for each block, whereas the query time for the other methods are completed in the space of few seconds. The table also illustrates that the 3D LBP approach outperforms the other three with sub-second retrieval time and the highest precision rate of 78%, as given in Table 3. However, this conclusion is very much content-based. In this case, the retrieval performance is based on the retrieval of images with similar lesion positions. Further studies are in need to explore whether any other contents, such as tumour shape, might be in favour of any of the other methods. All in all, all these four methods are implemented in the developed CBIR system that is addressed below.

IV. INTERFACE DESIGN

An online CBIR system, MIRAGE, acronym for Middlesex medical Image Repository with CBIR ArchivG Environment, for both 2D and 3D images has been developed and is online at [26]. Figure 9 demonstrates the interface of the system, whilst Figure 10 illustrates the flowchart of the architecture of interface. It consists of three modules with components of image classification, 2D image retrieval and 3D image retrieval respectively.

In Figure 9, the top picture displays a random selection of ten images from the collection of 'dateset_3' chosen from the dropdown menu of Collection Category, which can be achieved by simply pressing the 'Random' button. The last button on this figure gives the choice of the number of images to be displayed, which can be up to 140. Obviously more images will take longer to show up. Upon these shown images, users can pick one or more as query image or images by changing the status of each one from 'neutral' to 'rel' that refers to relevant, or 'non-rel' to eliminate the like of that image. By clicking the 'Query' button, the screen will show the retrieved images that are similar to the chosen query image or images.



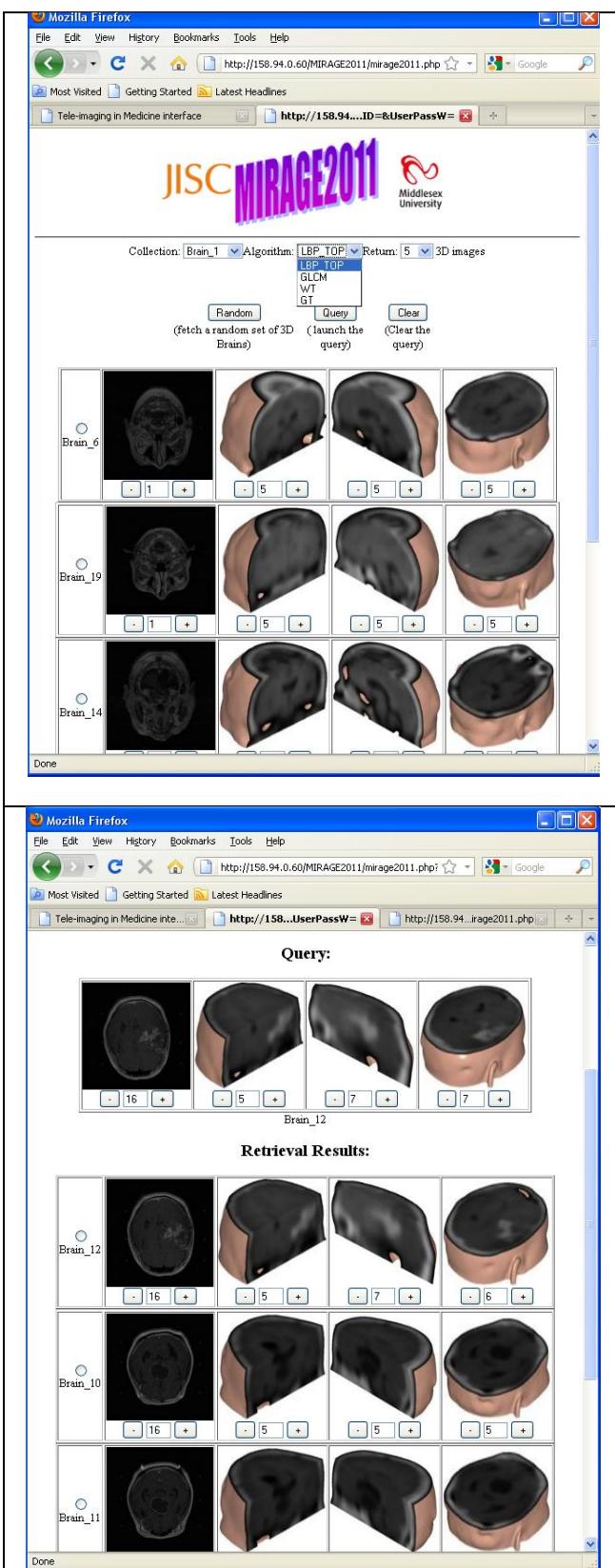


Figure 9. The interface of MIRAGE. Top: 2D images; Middle: 3D images; Bottom: retrieved results for 3D query.

The novelty of this work is the implementation of retrieval for 3D images which are demonstrated in the middle and bottom figures of Figure 9. The middle picture illustrates the

implementation of the four aforementioned algorithms that can be applied to the retrieval process. There are four ways to view each 3D dataset. Among the 5 columns in the figure, on the second left (the leftmost column lists the name of the data), 3D data are shown in 2D form. By clicking the '-' or '+' button at the bottom, users can view the 3D brain images slice by slice from the top of the head to the neck. In order to refer each slice to the 3D brain, the three columns on the right hand side showcase the mapping from 2D to 3D in the direction of back-front (coronal, column 3), left-right (sagittal, column 4), and top-bottom (axial, column 5). Similar to the 2D form of the top figure, a query image can be selected by ticking the image name and then pressing the 'Query' button. The retrieved images are then given as demonstrated in the bottom figure of Figure 9. Again, with the consideration of speed, only 5 datasets are shown at each time.

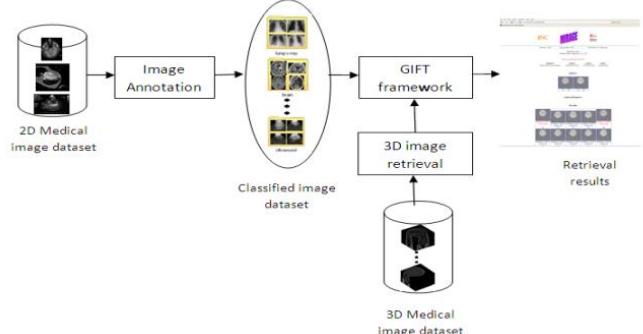


Figure 10. The framework for MIRAGE.

Built on the open source GNU Image Finding Tool (GIFT [40]), the online database is based on the Query-by-Example (QBE) paradigm coupled with a facility of user-relevance feedback whereby retrieved images most closely resemble a query image in appearance (i.e., the content that an image is carrying).

For 2D images, two algorithms have been implemented for indexing image collections, which are IDF (Inverse Document Frequency) and Separate Normalisation. The IDF is a classical method and is based on counting the number of documents in the collection being searched, which contain (or are indexed by) the terms in question [41], and has been applied in text retrieval systems, giving rise to the efficiency when employed in an image system. Conversely, feature normalisation refers to the compensation of scale disparity between the feature components that are defined in different domains.

On the client side, a web page based interface is given. Whilst the client-server communication is achieved using the XML-based Multimedia Retrieval Markup Language (MRML). All client-server communication, including queries from the client or results returned by the server, is realized through message passing. Consequently, the client can be implemented in any programming language. The current MIRAGE client is implemented using PHP (Personal Home Programming) language to generate dynamic web pages for the client web browser.

With respect to 3D interface, Figure 11 schematically illustrates a flowchart of the development.

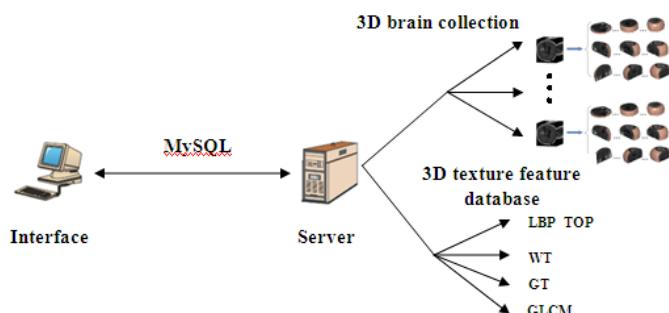


Figure 11. Framework of 3D brain retrieval system.

As illustrated in Figure 11, the visualization of 3D images relies on a Client-Sever architecture with MySQL communication protocol.

In order to display 3D brain as a whole instead of a pile of 2D slices, the skull of a brain is generated first from 3D volume data by using the method of iso-surface extraction, which is then followed by setting the step forward or backward with end-caps in three directions (i.e., X, Y, Z) respectively to show the inside structure of a brain from bottom to top, back to front and left to right, as schematically illustrated in Figure 12. All processing procedures including the step of images of 3D intersection (to be controlled by ‘-’ and ‘+’ buttons in Figure 9) and the extraction of texture features from aforementioned four approaches can be performed offline in advance. In this way, the created 3D inter-sections of brains with its original 3D brain image are then stored in the image database in a server, whilst texture features are stored in a feature database.

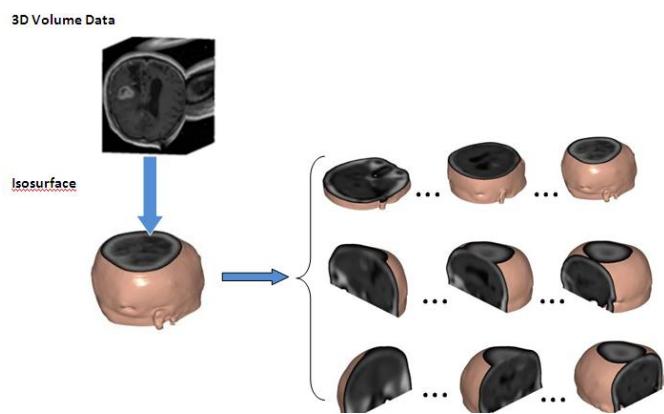


Figure 12 Creation of iso-surface and inter-sections of 3D volume data.

On the client side, interface relies on HTML, which is dynamically generated by means of hypertext preprocessor and therefore can be displayed in any internet browser. As shown in Figure 9, the user can view not only the original 3D brain image shown in first column of each row but also its cutaway in three different directions on the basis of slice by slice.

V. CONCLUSION

507

This paper is an extended version of [24], which introduces an online image retrieval system, MIRAGE, with a facility of CBIR for 3D images. Four texture based approaches that draw on the techniques of Local Binary Pattern, Grey Level Co-occurrence Matrices, Wavelet Transforms and Gabor Transforms have been exploited through the extension into 3D format, to retrieve lesioned MR brain images in this system. The results are very encouraging showing that not only higher precision rates can be achieved, but also that can it be done in real time. In comparison with each other, LPB outperforms the other three to a great extent whereas the 3D wavelet approach also performs well with similar retrieval accuracy, although slightly under-performed in terms of time. In terms of processing speed, it appears the pre-processing stage of detection of potential VOIs is essential to highlight lesions, the regions of interest that retrieved images should contain.

Because of the time required in the establishment of a feature database in 3D form, i.e., normalization, feature extraction, etc., in particular by using the approach of 3D GT (up to several minutes are needed for each brain), only ~100 datasets are included in this study. The very next step is to process more datasets. In addition, although the precision rate of 78% is very promising, a better rate may be possible by the combination of a few of these texture descriptors, while maintaining the short processing time. Comparison with shape based approaches is also in the pipeline, with the aim of developing CBIR systems for higher dimensional datasets.

ACKNOWLEDGMENT

This research is financially funded by UK JISC. Their support is gratefully acknowledged.

REFERENCES

- [1] Smeulders A.W., Worring M., Santini S., Gupta A., and Jain R., Content-based image retrieval at the end of the early years, *IEEE Trans. Pattern Anal. Mach. Intell.* 2000, 22 (12): 1349–1380.
- [2] Datta R., Joshi D., Li J., and Wang J., Image retrieval: ideas, influences, and trends of the new age, *ACM Computing Surveys*, 2008, 40(2), pp.5-1:60.
- [3] Novotni M. and Klein R., 3D Zernike Descriptors for Content Based Shape Retrieval, *Proceedings of the 8th ACM Symposium on Solid Modelling and Applications*, Seattle, Washington, USA, 2003, pp. 216-225.
- [4] Bustos B., Keim D., Saupe D., and Schreck T., Content-based 3D Object Retrieval, *IEEE Transactions on Computer Graphics and Applications*, 2007, 27 (4), pp.22-27.
- [5] Mademlis A., Darasb P., Tzovarash D., and Strintzis M.G., 3D Object Retrieval Using the 3D Shape Impact Descriptor, *Journal of Pattern Recognition*, 2009, 42 (11), pp.2447-2459

- [6] Cao L., Liu J., and Tang X., 3D Object Retrieval Using 2D Line Drawing and Graph Based Relevance Feedback, *Proceedings of the 14th Annual ACM International Conference on Multimedia*, Santa Barbara, CA, USA, 2006, pp. 105 – 108.
- [7] Ichida H., Itoh Y., Kitamura,Y., and Kishino F., Interactive Retrieval of 3D Shape Models Using Physical Objects, *Proceedings of the 12th Annual ACM International Conference on Multimedia*, New York, NY, USA, 2004, pp. 692 – 699.
- [8] Gong B., Xu C., Liu J., and Tang X., Boosting 3D Object Retrieval by Object Flexibility, *Proceedings of the 7th ACM International Conference on Multimedia*, Beijing, China, 2009, pp. 525-528.
- [9] Bustos B., Keim D., Saupe D., and Schreck T., Content-Based 3D Object Retrieval, *IEEE Computer graphics and Applications*, 2007, 27(4), pp. 22-27.
- [10] Vajramushti N., Kakadiaris I.A., Theoharis T., and Papaioannou G., Efficient 3D Object Retrieval Using Depth Images, *Proceedings of the 6th ACM SIGMM International Workshop on Multimedia Information Retrieval*, New York, NY, USA, 2004, pp. 189 – 196.
- [11] Lowe D. G., Distinctive Image Features from Scale-Invariant Keypoints, *International Journal of Computer Vision*, 2004, 60(2), pp. 91-110.
- [12] Scovanner P., Ali S., and Shah M., A 3-Dimensional SIFT Descriptor and Its Application to Action Recognition. *ACM Conference on Multimedia*, 2007, pp. 357-360.
- [13] Flitton G., Breckon T., and Megherbi N., Object Recognition using 3D SIFT in Complex CT Volumes, *British Machine Vision Conference (BMVC)*, 2010, pp.1-12.
- [14] Gao X. W., Qian Y., and Hui R., The state of the art of medical imaging technology: from creation to archive and back, *The Open Medical Informatics Journal*, 2011, 5 (Suppl 1), pp.73-85.
- [15] Arya M., Cody W., Faloutsos C., Richardson J., and Toya J., QBISM: Extending a DBMS to Support 3D Medical Images, *Proceedings of the Tenth International Conference on Data Engineering*, 1994, pp.314-325.
- [16] Liu Y. and Dellaert F., A classification based similarity metric for 3D image retrieval, Computer Vision and Pattern Recognition, *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'98)*, 1998, pp.800-805.
- [17] Chbeir R., Amghar Y., and Flory A., MIMS: a Prototype for medical image retrieval, *In RIAOCID*, 2000, pp.846-861.
- [18] Chu W., Hsu C., Cardenas A., and Taira R., Knowledge-based image retrieval with spatial and temporal constructs, *IEEE Transactions on Knowledge and Data Engineering*, 1998, 10(6), pp. 872–888.
- [19] Mojsilovic A. and Gomes J., Semantic based categorization, browsing and retrieval in medical image databases, *Proceedings of image Processing*, 2002, pp.III:145-148.
- [20] Glatard T., Montagnat J., and Mgann I.E., Texture based medical image indexing and retrieval: application to cardiac imaging, *Proceedings of the 6th ACM SIGMM international workshop on Multimedia information retrieval*, 2004, pp.135-142.
- [21] Cai W. and Feng D., Content-based Retrieval of dynamic PET functional images, *IEEE Transactions on Information Technology in Biomedicine* 2000, 4(2), pp.152-158.
- [22] Batty S., Fryer T., Clark J., Turkheimer F., and Gao X.W., Extraction of Physiological Information from 3D PET Brain Images, *VIIP'2002 (Visualization, Imaging and Image Processing)* 2002, pp. 401-405.
- [23] Gao, X.W., Qian Y., Loomes M., Comley R., Barn B., Chapman A., and Rix J., Texture-based 3D image retrieval for medical applications, *IADIS e-Health2010*, Germany, 2010, pp 29-31.
- [24] Qian Y., Gao X., Loomes M., Comley R., Barn B., Hui R., and Tian Z., Content-based retrieval of 3D medical images, *The Third International Conference on eHealth, Telemedicine, and Social Medicine, eTELEMED 2011*, France, IARIA, XPS Press, 2011, pp.7-12.
- [25] Talairach J. and Tournoux P., Co-planar stereotactic atlas of the human brain, Thieme, Stuttgart, 1988.
- [26] <http://image.mdx.ac.uk/vin/demo.php>. Retrieved on 26/1/2012.
- [27] Unay D., Ekin A., and Jasinschi R.S., Medical Image Search and Retrieval using Local Patterns and Kit Feature Points, *Proceedings of the International Conference on Image Processing*, San Diego, California, USA, 2008, pp. 997-1000.
- [28] <http://www.fil.ion.ucl.ac.uk/spm/>. Retrieved on 26/1/2012.
- [29] Montreal Neurological Institute, <http://www.mni.mcgill.ca/>. Retrieved on 26/1/2012.
- [30] Zhao G. and Pietikainen M., Dynamic Texture Recognition Using Local Binary Patterns with an Application to Facial Expressions, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2007, 9 (6) , pp. 915-928.
- [31] Gao, X.W., Batty, S., Clark, J., Fryer, T., and Blandford, A., Extraction of Sagittal Symmetry Planes from PET Images, *Proceedings of the IASTED International Conference on Visualization, Imaging, and Image Processing (VIIP'2001)*, 2001, pp. 428-433.
- [32] Bhattacharya A., On a Measure of Divergence between Two Statistical Populations Defined by Their Probability Distribution, *Bulletin of the Calcutta Mathematical Society*, 1943, 35, pp. 99-109.
- [33] Kovalev V.A., Kruggel F., Gertz F.J., and Cramon D. Y., Three-Dimension Texture Analysis of MRI Brain Datasets, *IEEE Transactions on Medical Imaging*, 2001, 20 (5), pp. 424-433.
- [34] Philips C., Li D., Raicu D., and Furst J., Directional Invariance of Co-occurrence Matrices within the Liver, *Proceedings of IEEE International Conference on Biocomputation, Bioinformatics, and Biomedical Technologies*, Bucharest, Romania, 2008, pp.29-34.

- [35] Haralick R.M., Shanmugam K., and Dinstein I., Textural Features for Image Classification, *IEEE Transactions on Systems, Man, and Cybernetics*, 1973, 3 (6), pp.610-621.
- [36] Mallat S. G., A Theory for Multiresolution Signal Decomposition: the Wavelet Representation, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1989, 11 (7), pp. 674-693.
- [37] Feng M. and Reed T.R., Motion Estimation in the 3-D Gabor Domain, *IEEE Transactions on Image Processing*, 2007, 16 (8), pp.2038-2047.
- [38] Wang Y. and Chua C., Face Recognition from 2D and 3D Images Using 3D Gabor Filters, *Journal of Image and Vision Computing*, 2005, 23 (11), pp. 1018-1028.
- [39] www.slicer.org. Retrieved on 26/1/2012.
- [40] <http://www.gnu.org/software/gift/>. Retrieved on 6/1/2012.
- [41] Robertson S., Understanding Inverse Document Frequency: On theoretical arguments for IDF, *Journal of Documentation*, 2004, 60 (5), pp.503–520.

Ontology Structure, Reasoning Approach and Querying Mechanism in a Semantic-Enabled Efficient and Scalable Retrieval of Experts

Witold Abramowicz, Elżbieta Bukowska, Monika Kaczmarek, Monika Starzecka

Department of Information Systems, Faculty of Informatics and Electronic Commerce, Poznań University of Economics,
Poznań, Poland

{w.abramowicz; e.bukowska; m.kaczmarek; m.starzecka}@kie.ue.poznan.pl

Abstract—Efficient utilization of knowledge became a key to the success of an organization. The need to identify experts within or outside an organization has been for a long time inspiration for various initiatives undertaken by academia and industry. The eXtraSpec system developed in Poland is an example of such initiatives. In order to realize its tasks, the eXtraSpec system needs not only to be able to acquire and extract information from various sources, but also requires an appropriate representation of information, supporting reasoning over person's characteristics. The considered mechanism should allow for a precise identification of required data, but simultaneously, be efficient and scalable. The main goal of this paper, is to present the ontology structure, reasoning approach as well as querying mechanism applied in the eXtraSpec project, and discuss the underlying motivation, which led to the development of a semantic-based mechanism to retrieve experts in its current state.

Keywords - Expert finding system; knowledge representation; expert characteristic, reasoning, querying

I. INTRODUCTION

An efficient acquisition and utilisation of knowledge is considered to be a key element contributing to a success of an organization operating in the competitive settings of the knowledge-based economy [49]. The organizations need not only to know the skills and expertise of their employees, but also need to be able to conduct an appropriate recruitment process. More and more often organizations, in order to locate expertise they require, take advantage of various Internet portals, including social portals, as well as other artefacts available on the Internet [50]. As the data and information on various experts available on WWW is very dispersed and of distributed nature, a need appears to support the processes of human resources management using the IT-based solutions e.g., information extraction and retrieval systems, especially expert retrieval systems.

Within an information retrieval (IR) process, a single query is executed on a set of documents in order to identify the relevant ones [2]. In general, a typical retrieval system encompasses three main components:

- a module responsible for collecting data (documents) and creating their easily processable representation in the form of an index;
- an interface allowing formulating queries reflecting the current information needs of a user and usually consisting of a set of keywords and finally,

- a mechanism matching a query to created indexes in order to identify the relevant documents.

All three elements affect the quality of the retrieval process, i.e., values of precision and recall metrics.

The traditional expert retrieval systems, being a subset of information retrieval systems focusing on identification of required experts, face the same problems as traditional IR systems. The mentioned problems are caused by usage of different keywords and different levels of abstraction by users when formulating queries on the same topic, or by using different words and phrases in the description of a phenomenon based on which indexes are created. In order to address these issues, very often semantics is applied, so as in response to a user query, a retrieval system returns documents, which do not contain words included in the query, but are still relevant to the user's information needs.

There are many research and commercial initiatives aiming at the development of retrieval systems in general and expert retrieval systems in particular, supported by semantics. They are to provide interested parties with detailed information on people's experience and skills. One of such initiatives is the Polish project eXtraSpec [23]. Its main goal is to combine company's internal electronic documents and information sources available on the Internet in order to provide an effective method of searching experts with competencies in the given field.

The main process in the eXtraSpec system flows as follows: the system acquires data from dedicated sources (on the Web or from the inside of the company) and saves it as an extracted profile (PE), whose structure is based on the European Curriculum Vitae Standard [38]. In the next step, data in PE is normalized. As a result of the normalization process, the normalized profile is generated (PN). Finally, PN are analysed and aggregated to the form of aggregated profile (PA) (i.e., one person is described by one and only one PA) serving as a source of information on experts. Based on the information provided by the aggregated profiles, the eXtraSpec system is to support three main scenarios:

- finding experts with desired characteristic,
- defining teams of experts, and
- verifying data on a person in question.

In order to support the above mentioned flow as well as three scenarios identified, the eXtraSpec system needs not only to be able to acquire and extract information from various sources, but also requires an appropriate representation of information that would support reasoning over person's characteristics. In addition, the reasoning

and querying mechanism should on the one hand, precisely identify required data, and, on the other, be efficient and scalable.

The main goal of this paper, being an extended version of [1], is to present more in depth the ontology structure, reasoning approach as well as querying mechanism applied in the eXtraSpec project, and discuss the underlying motivation, which led to the development of a semantic-based mechanism to retrieve experts in its current state.

In order to fulfill the mentioned goals, the paper is structured as follows. First, the related work in the area of expert's retrieval and using semantics to describe experts is discussed. Then, the description of the identified querying strategies constituting requirements for the defined solution follows. Next, the ontology developed for the needs of the eXtraSpec project to support retrieval of experts is presented. Then, the short description of the considered scenarios regarding the application of the reasoning infrastructure, as well as the description of the selected one, follows. Finally, the system architecture as well as implementation details of the reasoning mechanisms are given. The paper concludes with final remarks.

II. RELATED WORK

The need to find expertise within an organization has been for a long time inspiration for initiatives aiming at the development of a class of search engines, being a subset of information retrieval systems, called expert finders or expert retrieval systems [3].

There are several aspects connected with the expert finding task, for instance, following McDonald and Ackerman [4], those may be:

- expertise identification aiming at answering a question - who is an expert on a given topic?, and
- expertise selection aiming at answering a question - what does X know?

Within our research, we focus on the first aspect i.e., identifying a relevant person given a concrete need.

First systems focusing on the expertise identification task relied on a database like structure containing a description of experts' skills (e.g., [5]). However, such systems faced many problems, e.g.:

- how to ensure precise results given a generic description of expertise and simultaneously fine-grained and specific queries [6], or
- how to guarantee the accuracy and validity of stored information given the static nature of a database and volatile nature of person's characteristics.

To address these and similar problems other systems were proposed focusing on automated discovery of up-to-date information from specific sources such as e.g., e-mail communication [7]. In addition, instead of focusing only on specific document types, systems that index and mine published intranet documents [8] or analyse social networks [45], were proposed. An example may be the

Spree project [9] aiming at providing automatic expert finding facility, able to answer a given question. The system automatically builds qualification profiles from documents and uses communities and the social software in order to provide efficient searching capabilities.

In addition, currently the Web itself offers many other possibilities to find information on experts, as there are a number of contact management portals or social portals, where users can search for experts, potential employees or publish their curricula in order to be found by future employers (e.g., [25][26][27]).

When it comes to the algorithms applied to assess whether a given person is suitable to carry out a given task, at first, standard information retrieval techniques to locate an expert on a given topic were applied [10][11]. Usually, expertise of a person was represented in a form of a term vector and a query result was represented as a list of relevant persons.

If matching a query to a document relies on a simple mechanism checking whether a document contains the given keywords, then the well-known IR problems occur:

- low precision of returned results (there is a word, but not in this context);
- low value of recall (relevant documents described using a different set of keywords, are not identified);
- a large number of documents returned by the system (especially in a response to a general query) the processing of which is impossible (e.g., due to the time constraints).

Therefore, a few years ago, the Enterprise Track at the Text Retrieval Conference (TREC) was started in order to study the expert-finding topic. It resulted in further advancements of the expert finding techniques and application of numerous methods, such as probabilistic techniques or language analysis techniques to improve the quality of finding systems (e.g., [12][13][14][15]).

As the Semantic Web technology [42] is getting more and more popular [43], it is not surprising that it has been used to enrich descriptions within expert finding systems. The introduction of semantics into search systems may take two forms:

- the use of semantics in order to analyze indexed documents or queries (query expansion [44]),
- operating on semantically described resources with use of reasoners (e.g., operating on contents of RDF (Resource Description Framework [46]) files and ontologies represented in e.g., the OWL (Web Ontology language [47])).

Within the expert finding systems both approaches have been applied, as well as a number of various ontologies used to represent competencies and skills were developed.

For instance, the goal of a Single European Employment Market-Place (SEEMP) [16] was to provide interoperable architecture for e-Employment services. The mentioned project used an ontology in order to provide a semantic description of job offers and people's CV. The

main ontology developed within this project is called Reference Ontology and it consists of thirteen sub-ontologies: Competence, Compensation, Driving License, Economic Activity, Education, Geography, Job Offer, Job Seeker, Labour Regulatory, Language, Occupation, Skill and Time. The Reference Ontology has been built based on the commonly used standards, e.g., ISO 4217 [28], ISCO-88 COM [29], ONET [30] or DAML ontology [31].

In turn, in [17] authors describe requirements and a process of ontology creation for the needs of human resources management. They developed an ontology that is used in two projects: a meta-search engine for searching jobs on job portals [18] and by a university competence management system [19]. The ontology was created in the OWL formalism. It consists of sub-ontologies for competencies, occupations and learning objects.

Another example is the ExpertFinder system [20] being a framework for reuse of already existing vocabularies in order to apply them in semantically supported systems. It provides terms and best practices for describing web pages, persons, institutions, events, areas of expertise, relations between persons, educational aspects etc. ExpertFinder uses such vocabularies as: FOAF (Friend of a Friend) [32], SIOC [33], vCard [34] or Dublin Core [35].

In addition, numerous ontologies, taxonomies and classifications have been created in the human resources management area, e.g., taxonomies for job descriptions such as e.g., the Standard Occupational Classification (SOC) [36] of the United States Federal statistical agencies or taxonomy of skills developed within the KOWIEN project [21].

The problem tackled within this paper is related to the semantic-based expert finding. The eXtraSpec system acquires information from outside and assumes that one can build a profile of a person based on the gathered information. It is important for the users of an expert finding system that the system operates on a large set of experts. More experts imply bigger topic coverage and increased probability of a question being answered. However, it simultaneously causes problems connected to the heterogeneity of information as well as low values of both precision and recall of the system. The application of semantics may help to normalize the gathered data and ensure an appropriate level of precision and recall, however, it generates problems with scalability and efficiency of the designed mechanisms that need to be addressed.

When it comes to the ontology, the eXtraSpec system differs from other projects under a few aspects:

- it is not limited only to hierarchical relations;
- it has been developed for the Polish language and relates to Polish standards;
- it has been built in accordance to the Simple Knowledge Organization System (SKOS) [37] standard.

Applying semantics undoubtedly offers a way to handle the precision, recall, and helps to normalize data, however, the application of semantics impacts the performance as well as scalability of the system.

Therefore, a design decision needed to be taken regarding the way the semantics should be applied in order to ensure the required quality of the system. In the next section, we present the considered querying strategies, developed ontology, reasoning scenarios and the underlying motivation.

III. QUERYING STRATEGIES

In order to identify the requirements towards the persons' characteristics, scope of information needed to be covered by ontologies, as well as the querying and reasoning mechanism developed within the eXtraSpec system, first, exemplary searching strategies a user looking for experts may be interested in were considered. The strategies have been specified based on the conducted studies of the literature and interviews with employers conducting recruitment processes. The six most common searching goals are as follows:

1. To find an expert with some experience at a position/role of interest.
2. To find an expert having some specific language skills on a desired level.
3. To find an expert having some desired competencies.
4. To find students who graduated recently/will graduate soon in a given domain of interest.
5. To find a person having expertise in a specific domain.
6. To find a person with specific education background, competencies, fulfilled roles, etc. Although the enumerated goals (1-5) sometimes are used separately, usually though, they constitute building blocks of more complex scenarios within which they are freely combined using various logical operators.

As already mentioned, the above querying goals imposed some requirements on the information on experts that should be available, and in consequence, also ontologies that needed to be developed for the project's needs, as well as the reasoning and querying mechanism. Tables 1-3 summarize the requirements on the scope of information required to describe an expert, on querying and reasoning mechanism, as well as on the ontology itself.

TABLE 1 QUERYING STRATEGIES AND RESULTING REQUIREMENTS ON THE SCOPE OF INFORMATION

Scenario No.	Requirements on the scope of information
1. To find an expert with some experience on a position of interest.	An expert description MUST include information on positions and jobs undertaken so far as well as their duration.
2 To find an expert having some specific language skills on a desired level.	An expert description MUST provide information on: known languages, obtained certificates and a level of language skills.
3 To find an expert having some competencies.	An expert description MUST include information on soft and tangible competencies of a person.

Scenario No.	Requirements on the scope of information
4 To find students who graduated recently or will graduate soon in a given domain.	An expert description MUST include information on educational background of a person, especially: educational organization, date of graduation and educational result.
5 To find a person having expertise in a specific domain.	An expert description MUST include information on organizations a person worked for. Please note that the information on the domains the organizations operate in should be provided by ontology (see Table 3)
6 To find a person with specific education, competencies, jobs, etc.	Features of interests for this scenario include all previously mentioned.

TABLE 2 QUERYING STRATEGIES AND RESULTING REQUIREMENTS ON REASONING AND QUERYING MECHANISM

Scenario No.	Requirements on ontology
3 To find an expert having some competencies	The ontology MUST represent skills and competencies and their hierarchical dependencies as well as some additional relations as appropriate.
4 To find students who graduated recently/will graduate in a given domain	The ontology MUST provide a hierarchy of educational organizations allowing for their categorization and reasoning on their hierarchical dependencies.
5 To find a person having expertise in a specific domain	The ontology SHOULD provide information on organizations allowing for their categorization (is-a relation) as well as provide information on the domains they operate in.
6 To find a person with specific education, competencies, jobs, etc.	Requirements on ontologies are the same as in scenarios 1-5.

The next section presents the developed ontology meeting the above enumerated requirements.

IV. ONTOLOGIES IN THE EXTRASPEC PROJECT

A. Requirements

The ontology developed for the system, as already mentioned in the previous section, needed to support the defined requirements resulting from the identified strategies. However, also some additional requirements, resulting from the already presented system flow, have been identified.

The eXtraSpec system acquires automatically data from dedicated sources, both company external and internal ones. The extracted content is saved as an extracted profile (PE), which is an XML file compliant with the defined structure of an expert profile based on the European Curriculum Vitae Standard [38]. Therefore, it consists of a number of attributes, such as e.g., education level, position, skill, that are assigned to different profile's categories such as e.g., personal data, educational history, professional experience. Vocabulary in the extracted content is then processed and normalized using the developed ontology. The result of the normalization process is a normalized profile (PN). An important assumption is: one standardized profile describes one person, but one person may be described by a number of standardized profiles (e.g., information on a given person at different points of time or information acquired from different sources). Thus, normalized profiles are analysed and then aggregated, in order to create an aggregated profile (PA) of a person. Finally, the reasoning mechanism is fed with the created aggregated profiles and answers user queries on experts. Thus, the additional requirements the ontology should address are as follows:

1. The ontology MUST enable semantic annotation of all elements of aggregated profile.
2. The ontology MUST support the normalization process of extracted profiles.

The creation of ontology for the needs of the eXtraSpec project was preceded by thorough analysis of

TABLE 3 QUERYING STRATEGIES AND RESULTING REQUIREMENTS ON ONTOLOGY

Scenario No.	Requirements on ontology
1. To find an expert with some experience on a position of interest.	The ontology MUST represent a is-a hierarchy of different positions and jobs allowing for their categorization and reasoning on their hierarchical relations.
2 To find an expert having some specific language skills on a desired level.	The ontology MUST represent languages certificates (is-a hierarchy) together with information on the language and the proficiency level, mapped to one scale.

the requirements resulting from the scenarios supported by the system as well as those mentioned above. In addition, the consequences of applying various formalisms and data models for the ontology modelling, and its further application, were investigated. In consequence, three assumptions were formulated:

- only few relations will be needed and thus, represented,
- developed ontologies should be easy to translate into other formalisms,
- the expressiveness of used ontology language is important, however, the efficiency of the reasoning mechanism is also crucial.

B. Formalism

As the result of the conducted analysis of different formalisms and data models, the decision was taken to apply the OWL language as the underlying formalism and the Simple Knowledge Organization System (SKOS) [37] model as a data model. The criteria that influenced our choice were as follows:

- relatively easy translation into other formalisms;
- simplicity of representation;
- expressiveness of used ontology language;
- efficiency of the reasoning mechanism.

Many knowledge representations, such as thesauri, taxonomies and classifications, share some structure elements and are used in similar applications. SKOS gathers most of those similarities and explicitly enables data and technology exchange between different applications. The SKOS data model enables low cost migration that allows making a connection between existing SKOS and the Semantic Web. Ontologies developed in accordance to the SKOS model can be expressed in any known ontology language.

Because of the strong software support and a wide usage of OWL, we decided to use that formalism within our work.

C. Model

The basic element of the eXtraSpec system is an already mentioned profile of an expert. Each expert is described with series of information, for example: name and family name, history of education, career history, hobby, skills, and obtained certificates. For the needs of the project, a data structure to hold all that information was designed. To make the reasoning possible, a domain knowledge for each of those attributes is needed. The domain knowledge is represented by the ontology. Ten attributes from the profile of an expert were selected to be a ‘dictionary reference’, i.e., the attributes, whose values are references to instances from the ontology. Those attributes are:

- educational organization – name of organization awarding a particular level of education or educational title;
- certifying organization – name of an organization that issued the particular certificate;

- client, employer and role – those three attributes are used to describe the history of employment. A single step in the employment history is described as a business relation. Each relation consists of three basic elements: client (i.e., an employer) and a role (i.e., profession) that an expert fulfilled in this relation;
- scope of education – the domain of education (for example: IT, construction, transportation);
- topic of education – for a higher education description, it will be a name of the specialization, for trainings or courses etc. – their topic;
- result of education – the obtained title;
- skill – an ability to perform an activity or job well, especially because someone has practiced it;
- name of a certificate;
- degree of a skill.

Performed analysis of the requirements imposed on the ontology for the needs of reasoning, concluded with the definition of a set of relations that should be defined. They are as follows:

- hasSuperiorLevel – representing hierarchical relations between concepts,
- isEquivalent – representing the substitution between concepts,
- isLocatedIn – representing various geographical dependencies,
- isLocatedInCity – representing geographical dependencies,
- isLocatedInVoivodeship – representing geographical dependencies,
- provesSkillDegree – connecting skills and certificates,
- worksInLineOfBusiness – representing dependencies between organizations and lines of business,
- isPartOf – representing a composition of elements, for example: ability of using MSWord is a part of ability of using MSOffice (however, knowing MSWord does not imply that a person knows the entire MSOffice suit).

Additionally, various built-in SKOS relations have been used, namely:

- broader,
- hasTopConcept,
- inScheme,
- narrower,
- topConceptOf.

The SKOS model, while providing simplicity and easy translation into many different formalisms, imposes some restrictions. The most important one is the lack of support for some features and facilities provided by the OWL language. An overall idea of an ontology stack apart of concepts and data properties, assumed definition of some object properties. The designed ontology needed to be coherent with the SKOS model specification, processable by the used SKOS API and still represent all above

mentioned areas and relations. To meet all those assumptions, the designed data structure is one SKOS ontology with eight concept schemas for each area of interest: Organizations (for organizational organizations, certifying organizations, Employer and Client), SkillName, SkillDegree, Certificate, Role, EducationScope, EducationTopic, EducationResult as well as complementary schemas for Cities and Voivodeship, Languages and Line of Business.

In the process of profile normalization the values from the extracted profile are linked to the concepts from the ontology. It is possible that the normalization mechanism will not be able to find the extracted value within the ontology. In this case, we assume that the extracted value should not be discarded; instead, it should be added to the appropriate Concept Schema. Therefore, every Concept Schema has a top concept TMP. Possible candidates for new concepts are added as a subConceptOf TMP, and later can be resolved by an expert. In this way, we make it possible to extend the ontology with new concepts found in the Internet or other sources describing expert's profiles.

D. Sources of information

While building the ontology for the needs of the eXtraSpec system, a wide range of taxonomies and classifications has been analyzed in order to identify the best practices and solutions. As the eXtraSpec system is a solution designed for the Polish language, so is also the developed ontology. In order to develop particular Concept Schemas information from series of sources was incorporated. Table 4 shows the exemplary sources used to create the ontology structure as well as instances of numerous concepts.

TABLE 4 SOURCES OF INFORMATION

Concept schema	Sources of information
Organizations	The branch with Educational Organizations currently includes all Polish academic organizations, according to the official list published by the Polish Ministry of Science and Higher Education [39]. Additionally, a branch with employer organizations has been prepared based on the publicly available Internet sources.
Role	As this concept schema includes the classification of legally named professions in Poland, the source in this case was the official Polish Classification of Occupations [40] published by the Polish Ministry of Labor and Social Policy.
EducationScope	The data to create this Concept Schema was obtained from a number of Polish online job portals. The list of topical areas of education was slightly different in every portal. The final list of concepts in EducationScope Concept Schema is a combination of all of them.

Concept schema	Sources of information
EducationTopic	Currently this concept schema includes a list of specializations that a student may graduate in at Polish Higher Education Organizations based on the official register published by the Polish Ministry of Science and Higher Education.
EducationResult	This concept schema includes scientific titles, occupational titles and academic degrees that may be obtained in Poland based on the appropriate ordinances of the Polish Ministry of Science and Higher Education.
CertificateName	On-going analysis focuses on language certificates possible to be obtained by Polish citizens.
SkillName	This concept schema was based on series of skill classifications provided by Polish job portals, as well as international scientific publications from the area of human resources management and IT solutions for human resources management area.
SkillDergee	On-going analysis focuses on solutions used in the mentioned job portals.
City	In this case the list of Polish cities was used.
Language	In this case a list of languages a good command of which can be proved by a certificate was utilised.
LineOfBusiness	In this case a list of lines of business that are used by job portals was prepared.
Voivodeship	In this case a list of Polish Voivodeships was used.

V. QUERYING AND REASONING MECHANISM

One of the most important functionalities of the eXtraSpec system is the identification of persons having the desired expertise. The application of the Semantic Web technologies in order to ensure the appropriate quality of returned results implies application of a reasoning mechanism to answer user queries.

In order to support the querying and reasoning scenarios, the eXtraSpec system needs not only an appropriate representation of information supporting reasoning over person's characteristics (as described within the previous section), but also the querying and reasoning mechanism itself supporting on the one hand, precise identification of required data, and on the other hand, being efficient and scalable.

A. Approaches to semantic-enabled reasoning

Given the above criteria (precision and recall on the one hand, and efficiency and scalability on the other), three possible approaches were considered.

The first approach involves using the fully-fledged semantics by expressing all expert profiles as instances of an ontology, formulating queries using the defined ontology, and then, executing a query using the reasoning mechanism. This approach involves the need to load all ontologies into the reasoning engine and representing all individual profiles as ontology instances. The performed experiments showed that querying the reasoning infrastructure, even while using only a small set of

gathered profiles, is a resource (large memory consumption) and time consuming task (up to a few minutes). Therefore, although having a high precision and recall, it has poor performance and scalability.

The *second* approach relies on the query expansion using an ontology, i.e., adding keywords to the query by using an ontology to narrow or broaden the meaning of the original query. It allows getting answers faster than the previous approach, however, it could not take into account additional relations expressed in the ontology, and therefore, did not always allow for an increased precision. In addition, each user query needs to be normalized and then expanded using the ontology, therefore, the application of a reasoner was necessary. The experiments showed that it affected the values of the system performance and scalability.

The *third* approach called pre-reasoning involves two independent processes:

- creation of enriched profiles (indexes), to which additional information reasoned from the ontology is added and saved within the repository as syntactic data;
- formulating a query with the help of the appropriate GUI using the defined ontology serving as a controlled vocabulary. Then, the query is executed directly on a set of profiles using the traditional mechanisms of IR. There is no need to use the reasoning engine while executing a query.

This approach allows circumventing the drawbacks associated with the first approach, shifting the burden of an operation on the stage of indexing using ontologies.

Our experiments proved that applying the fully-fledged semantics is a precise, but neither efficient nor scalable solution. The query expansion provides an increased precision of the results (in comparison to the traditional IR mechanisms) and has better scalability and efficiency than the fully-fledged semantics, however, does not allow to take full advantage of the developed ontologies and existing relations between concepts. Only application of the third considered approach allows taking advantage of the mature IR mechanisms while increasing the accuracy and completeness of the returned results by: introducing a preliminary stage called pre-reasoning in order to create enriched indexes and the minimum use of the reasoning engine during the search.

B. Querying and reasoning component – architecture

The eXtraSpec system consists of a number of modules specialized for different tasks. Its architecture is described in [23], in this paper we focus on the REA component (REASONing) presented in Figure 1.

REA consists of an indexing mechanism (indexer), a searching mechanism (searcher), a composition mechanism (composer) and a reasoning engine with a set of ontologies loaded.

The selected approach requires the support of two independent processes:

- First, creating indexes of profiles - optimized for search, i.e., structured so as to enable a very fast

search based on criteria pre-set by a user. The aggregated profile is analysed, divided into relevant sections, and then enriched with additional information using the ontology (pre-reasoning). Any modification of the ontology forces the need to change indexes.

- The second process that needs to be supported is defining the query matching mechanism on the enriched indexes - this process is initiated by the task of a user formulating queries using a graphical interface that is also discussed later within this section. An employer, constructing a query points to interesting criteria and values they should meet. In the background, the desired values of various features from the lists and combo boxes, point to specific elements from the ontology [48].

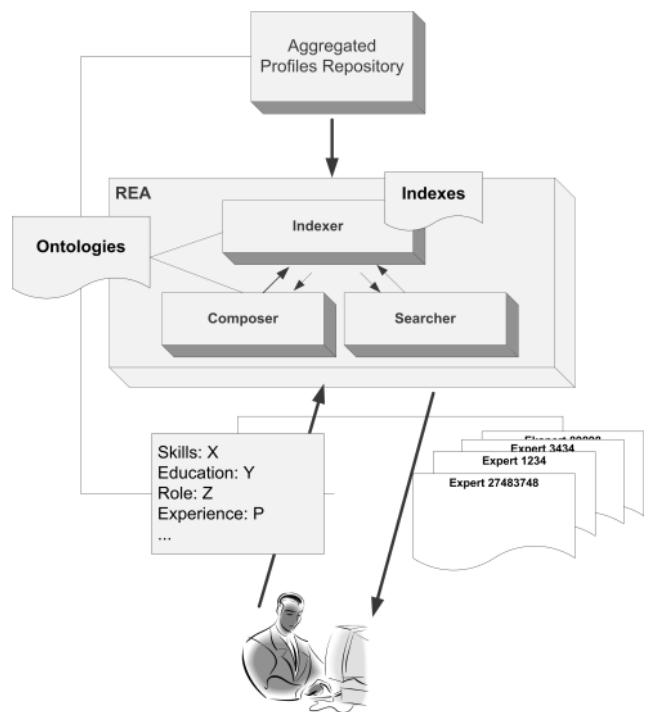


Figure 1. REA Overview

C. Profile structure

To realize the information retrieval side of the mechanism, the open-source java library Lucene [41], supported by the Apache Software Foundation, was selected. Instead of searching text documents directly, Lucene searches the previously prepared index. This speeds up the searching process and makes it more efficient. An index consists of at least one document. A document is a basic unit that is indexed and searchable, and represents text files, HTML code or database tables. A single document consists of fields. Each field has a unique

name (used as a key) and a value. The result of the search process is a list of all relevant documents.

Fields in the Lucene documents cannot be grouped together nor stored as hierarchical structures. However, within an aggregated profile (PA), which is a base profile for searching, some hierarchies and groups might be found. Since an explicit mapping from PA to the Lucene document is not possible, during the indexing process profiles are divided into a number of separate documents as also shown in Figure 2.

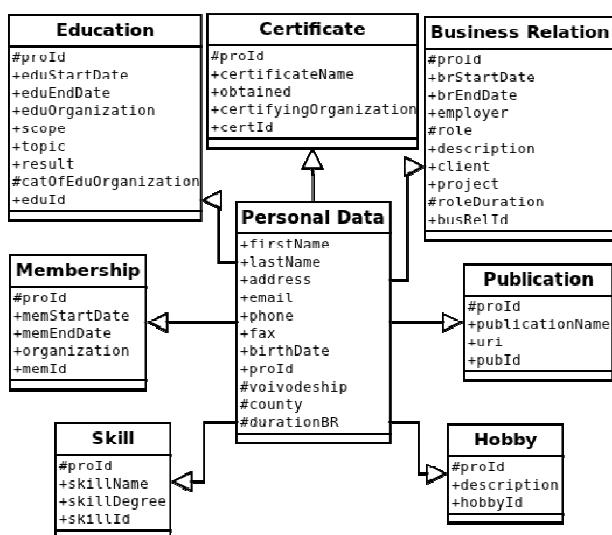


Figure 2. Data model overview

Each person is represented by exactly one Personal Data document and a number of corresponding documents that represent different groups of information. Each document contains an additional field with the profile ID that enables binding documents with the expert's main profile. Thus, for each listed category from PA, the separate Lucene document is created, e.g., for one obtained certificate, one document is created. The mentioned documents are as follows:

- personal data (e.g., first name, last name, phone number, address),
- history of education,
- certificates,
- skills,
- publications,
- mentions,
- history of employment,
- organisations,
- hobby.

Concurrently with the indexing process, pre-reasoning takes place, in order to enhance the profile with the implied facts. The documents contain fields generated directly from PA (marked with +) as well as additional fields (marked with #). Moreover, fields such as e.g., role,

skillName, catOfEduOrganization contain not only the concept from PA but also a hierarchy of its super-concepts from the ontology. Super-concepts are indexed as additional values for the given document field: these values are saved as next array elements and it is assumed that the higher array index number, the smaller weight the concept has. The assigned weight affects the ranking procedure.

As already mentioned, if the returned super-concepts do not correspond with the PA elements conceptually, additional fields are added to the document being indexed. For example, PA element 'address' might be divided into data that is more detailed, i.e., zip code, city, street, etc. Based on the zip code it is possible to specify the county and the province, and search for experts using the spatial criteria. Since PA does not contain such elements, we add fields to the personal data document during the indexing process.

D. Query structure

Lucene provides a very flexible but simple query structure. Therefore, in the eXtraSpec system it had to be extended in order to correspond to the defined requirements that result from the querying scenarios. They are as follows:

1. The querying and reasoning mechanism MUST allow building queries in a structured way (i.e., feature: desired value).

2. The querying and reasoning mechanism MUST support definition of desired values of attributes in a way suitable to the type of data stored within the given feature (i.e., text fields using wild-cards, date fields - after or before certain dates; numbers - less than..).

3. The querying and reasoning mechanism MUST allow to join a subset of selected criteria within the same category into one complex requirement (e.g., category: education; {education level: university AND finished date: after 2010 year}) using different logical operators.

4. The querying and reasoning mechanism MUST allow formulating a set of complex requirements within one category with different logical operators.

5. The querying and reasoning mechanism MUST allow joining complex requirements formulated in various profile categories into one criteria with different logical operators.

The logical operators between different sets of criteria and criteria themselves include such operators as: must, should, must not.

In order to answer more sophisticated queries encompassing several criteria from various documents, users' queries are executed on the index using a set of QueryObjects for different categories. Those QueryObjects are in turn sets of QueryObjects within the given category, each consisting of a set of QueryObject's structures consisting of a query string and a query operator. A query string is a Lucene compliant phrase that includes the field name and the relative value. A query operator is a logical operator: MUST, SHOULD, MUST_NOT, that defines

whether the specified criteria should be included or excluded from the result set.

The performed tests have shown that the defined query object fulfils the formulated requirements.

The application of semantics in the form of a pre-reasoning phase allowed achieving precise results, simultaneously allowing taking advantage of the matured IR mechanisms guaranteeing scalability and good performance of the system. Such a structure of the query together with the set of defined methods allow to address the scenarios defined above, however, makes formulating queries more complicated for users. Thus, a challenge of designing a user-friendly interface has appeared. The developed interface is shortly described in the next subsection.

E. GUI

The front-end to the eXtraSpec system should enable users to build complex queries describing characteristics of desired experts. During the analysis phase the main requirements for the system interface have been defined, namely [48]:

1. The interface MUST enable a user to specify constraints on expert's attributes and select whether the value of an attribute is required, desired (but not required) or not allowed.
2. The interface SHOULD enable grouping of constraints e.g., it should be possible to specify a graduated school and graduation date as one criterion.
3. The interface SHOULD provide a possibility to build queries which include complementary and alternative constraints.
4. The interface SHOULD enable providing some of criteria values typed-in as free text (with wildcards) and some of them to be selected from the eXtraSpec system knowledge base.
5. The interface SHOULD be loosely coupled with the system.
6. The interface SHOULD be understandable and easy to use.

The conceptual model of the interface is determined by the scheme of querying the experts finding system and the structure of the aggregated profile. The search criteria are divided into the following categories: personal data, education, professional experience, foreign languages, courses, certificates, additional skills, organization membership and interests.

Figure 3. The eXtraSpec GUI (1)

Categories consist of groups of fields. Desired values of these fields are specified in the interface by criteria values, and field groups by criteria groups. Each criterion has a label and a value typed by the user, selected from list or from values tree loaded from the ontology.

As a result eXtraSpec system front-end is a dynamic web user interface with cross-browser compatibility.

Figure 4. The eXtraSpec system GUI (2)

The developed interface has been successfully evaluated. See [48] for more details.

VI. CONCLUSIONS

The main goal of the eXtraSpec project is to develop a system supporting analysis of company documents and selected Internet sources for the needs of searching for

experts from a given field or with specific competencies. The provided system focuses on processing texts written in the Polish language. The obtained information is stored in the system in the form of experts' profiles and may be consolidated when needed. The system aims to offer a user friendly interface to perform queries that allow to find persons with specific characteristics. Realisation of this goal requires interconnection between the developed interface and underlying ontologies. Within this paper, we have discussed the concept and considered scenarios regarding the implementation of the querying and reasoning mechanism for the needs of the eXtraSpec system. We argue that by introducing the pre-reasoning phase, the application of semantics may be used to achieve precise results when searching for experts and at the same time, ensure the proper performance and scalability.

The set of developed ontologies discussed within this paper was designed specially for the Polish language, however, the main structure and model as well as defined relations may be reused also for other languages. The ontology in question is still under development, however, in the current state of affairs the reasoning about competencies in order to complete the expert profile with additional data on education, work experience is successfully performed by the REA component described within this paper. Our current work focuses on the implementation of the second scenario supported by the eXtraSpec system i.e., composition of teams of experts using the developed ontology.

ACKNOWLEDGMENT

The work published in this article was supported by the project titled: "Advanced data extraction methods for the needs of expert search" financed under the Innovative Economy Framework and partially supported by European Union in the European Regional Development Fund programme (contract no. UDA-POIG.01.03.01-30-150/08-01).

REFERENCES

- [1] Abramowicz, W., Bukowska, E., Kaczmarek, M., Starzecka, M. "Semantic-enabled Efficient and Scalable Retrieval of Experts", Proceedings of Third International Conference on Information, Process, and Knowledge Management (eKNOW), 2011
- [2] van Rijsbergen, C. J.; "Information Retrieval and Information Reasoning". Computer Science Today 1995, pages 549-559
- [3] Yimam, D.; "Expert finding systems for organizations: Domain analysis and the demoir approach" in: ECSCW 99 Workshop: Beyond KNowledge Management: Managing Expertise, pages 276–283, New York, NY, USA, 1996. ACM Press
- [4] McDonald, D. W. and Ackerman, M. S.; "Expertise recommender: a flexible recommendation system and architecture" in: CSCW '00: Proceedings of the 2000 ACM conference on Computer supported cooperative work, pages 231–240. ACM Press, 2000.
- [5] Yimam-Seid, D. and Kobsa, A. "Expert finding systems for organizations: Problem and domain analysis and the demoir approach". Journal of Organizational Computing and Electronic Commerce, 13(1):1–24, 2003
- [6] Kautz, H., Selman, B., and Milewski, A.; "Agent amplified communication" in: Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96), pages 3–9, 1996
- [7] Campbell, C. S., Maglio, P. P., Cozzi, A., and Dom, B.; "Expertise identification using email communications" in: CIKM '03: Proceedings of the twelfth international conference on Information and knowledge management, pages 528–531. ACM Press, 2003
- [8] Hawking, D.; "Challenges in enterprise search" in: Proceedings Fifteenth Australasian Database Conference, 2004
- [9] Metze, F., Bauckhage, Ch., and Alpcan, T., "The "Spree" Expert Finding System" in: Proceedings of the First IEEE International Conference on Semantic Computing (ICSC 2007), September 17-19, 2007, Irvine, California, USA
- [10] Ackerman, M.S., Wulf, V. and Pipek, V.; "Sharing Expertise: Beyond Knowledge Management"; MIT press, (2002).
- [11] Krulwich, B. and Burkey, C.; "ContactFinder agent: answering bulletin board questions with referrals" in: Proceedings of the National Conference on Artificial Intelligence, pages 10-15, 1996
- [12] Balog, K., Azzopardi L. and De. Rijke, M.; "Formal models for expert finding in enterprise corpora" in: Proceedings of the ACM SIGIR, pages. 43-50, 2006.
- [13] Fang, H. and Zhai, C.; "Probabilistic models for expert finding" in: Proceedings of the ECIR, pages 418-430, 2007
- [14] Petkova, D. and Croft, W.; "Hierarchical language models for expert finding in enterprise corpora" in: Proceedings of the 18th IEEE International Conference on Tools with Artificial Intelligence, pages 599-608, 2006
- [15] Serdyukov, P. and Hiemstra, D.; "Modeling documents as mixtures of persons for expert finding" in: Proceedings of the ECIR, pages 309-320, 2008.
- [16] Gómez-Pérez, A., Ramírez, J., and Villazón-Terrazas, B., "An Ontology for Modelling Human Resources Management Based on Standards" in: B. Apolloni et al. (Eds.): KES 2007/WIRN 2007, Part I, LNAI 4692, pp. 534–541, 2007
- [17] Dorn, J., Naz, T., and Pichlmair, M., "Ontology Development for Human Resource Management" in: "Proceedings of 4rd International Conference on Knowledge Management", Ch. Stary, F. Barachini, and S. Hawamdeh (Hrg.); Series on Information&Knowledge Management, 6 (2007), ISBN: 978-981-277-058-5; S. 109 - 120.
- [18] Dorn, J. and Naz, T.; "Meta-search in Human Resource Development", in: Proceedings of 4th Int. Conference on Knowledge Systems, Bangkok, Thailand, 2007
- [19] Dorn, J. and Pichlmair, M.; "A Competence Management System for Universities", in: European Conference on Information Systems, St. Gallen, 2007
- [20] Aleman-Meza, B., Bojars, U., Boley, H., Breslin, J.G., Mochol, M., Nixon, L.J.B., Polleres, A., and Zhdanova, A.V., "Combining RDF Vocabularies for Expert Finding"
- [21] Dittmann, L.; "Towards Ontology-based Skill Management, Projektbericht zum Verbundprojekt KOWIEN", Universität Duisburg-Essen, 2003.
- [22] Abramowicz, W., Wieloch, K.; "Raport podsumowujący wyniki prac przeprowadzonych w ramach zadań Z1.1, Z1.2 oraz Z2.1", Technical report of the eXtraSpec project, Department of Information Systems, Poznan University of Economics, 2009

- [23] Abramowicz, W., Kaczmarek, T., Stolarski, P., Węcel, K., and Wieloch, K.; "Architektura systemu wyszukiwania ekspertów eXtraSpec", in: Proceedings of "Technologie Wiedzy w Zarządzaniu Publicznym", Hucisko, 19-21 September 2010
- [24] <http://extraspeskie.poznan.pl/>, last access date: 20.01.2012
- [25] <http://www.bizwiz.com>, last access date: 20.01.2012
- [26] <http://www.xing.com>, last access date: 20.01.2012
- [27] <http://linkedin.com>, last access date: 20.01.2012
- [28] <http://www.iso.org/iso/en/prods-services/popsids/currencycodeslist.html>, last access date: 20.01.2012
- [29] http://ec.europa.eu/eurostat/ramon/index.cfm?TargetUrl=DS_PUB_WELC, last access date: 20.01.2012
- [30] <http://online.onetcenter.org/>, last access date: 20.01.2012
- [31] <http://cs.yale.edu/homes/dvm/daml/time-page.html>, last access date: 20.01.2012
- [32] <http://www.foaf-project.org/>, last access date: 20.01.2012
- [33] <http://sioc-project.org/>, last access date: 20.01.2012
- [34] <http://www.imc.org/pdi/>, last access date: 20.01.2012
- [35] <http://dublincore.org/>, last access date: 20.01.2012
- [36] <http://www.bls.gov/soc>, last access date: 20.01.2012
- [37] <http://www.w3.org/TR/swbp-skos-core-spec>, last access date: 20.01.2012
- [38] <http://www.europa-pages.com/jobs/europass.html>, last access date: 20.01.2012
- [39] <http://www.nauka.gov.pl/szkolnictwo-wyzsze/system-szkolnictwa-wyzszego/uczelnie/>, last access date: 20.01.2012
- [40] http://www.praca.gov.pl/pages/klasyfikacja_zawodow2.php, last access date: 20.01.2012
- [41] <http://lucene.apache.org>, last access date: 20.01.2012
- [42] Berners-Lee, T., Hendler, J. & Lassila, O., "The semantic web", Scientific American, May, 2001, pages 35-43.
- [43] Shadbolt, N.; Berners-Lee, T.; Hall, W., "The Semantic Web Revisited", IEEE Intelligent Systems Journal, Vol. 21, no. 3, 2006 page. 96-101.
- [44] Navigli, R., Velardi, P. "An Analysis of Ontology-based Query Expansion Strategies", Proceedings of Workshop on Adaptive Text Extraction and Mining at the 14th European Conference on Machine Learning, Cavtat-Dubrovnik, Croatia, 2003, pp. 42-49
- [45] Michalski, R., Palus, S., Kazienko, P., "Matching Organizational Structure and Social Network Extracted from Email Communication", Lecture Notes in Business Information Processing, 2011, Volume 87, Part 6, Part 6, 197-206
- [46] <http://www.w3.org/RDF/>, last access date: 20.01.2012
- [47] <http://www.w3.org/TR/2004/REC-owl-features-20040210/>, last access date: 20.01.2012
- [48] Abramowicz, W., Bukowska, E., Dzikowski, J., Filipowska, A., Kaczmarek, M., "Web Interface for Semantically Enabled Experts Finding System", in: ICEIS 2011, Proceedings of the 13th International Conference on Enterprise Information Systems , Beijing, SciTePress – Science and Technology Publications, 2011. pp. 291-296, ISBN 978-989-8425-56-0
- [49] Distribution, G., & Lundvall, B. A. "The Knowledge-based Economy", Development (96), 115.,OECD, 1996. <http://www.oecd.org/dataoecd/51/8/1913021.pdf>, last access date: 20.01.2012.
- [50] Chasins, Jeff. "Social media, recruiting, and job boards: which way are we going?" ere.net. 14 Sept. 2010. Ere Media, Inc. 22 December 2010, <http://community.ere.net/blogs/job-board-doctor/2010/09/social-media-recruiting-and-job-boards-whichway-are-we-going>, last access date: 20.01.2012

Block Matching Motion Estimation with Variable Search Window Size

Ionuț Pirnog and Claudia Cristina Oprea

Telecommunications Department

“Politehnica” University of Bucharest

313, Splaiul Independenței, Sector 6, 060042

Bucharest, Romania

ionut@comm.pub.ro, cristina@comm.pub.ro

Abstract— Block matching algorithms for motion estimation were developed in order to obtain reasonable motion estimation efficiency with minimum computational cost. Although the gain in the computational complexity is significant these algorithms have less precision in estimation than the basic block matching motion estimation algorithm, i.e., the Full Search algorithm. The proposed motion estimation method can be used with any of the existing block matching algorithms and brings an increase of estimation precision with small increase of the global computational cost. This improvement is achieved by choosing the search window size depending of the ration between the frame size and the motion area size.

Keywords – motion estimation, block matching, variable search window

I. INTRODUCTION

Motion information is very useful in the video compression process [2] since the development of video content retrieval applications [3]. In the video compression systems, the motion vectors are used for the representation of a video frame based on the previous frames. In the content retrieval systems, the video content can be found based on the video motion properties expressed by motion descriptors [4]. The extraction of motion vectors from a video frame based on the previous frame is known as motion estimation. There are many motion estimation methods, e.g., parametric methods, stochastic methods. The simplest and most used motion estimation method is the one based on block matching. The block matching algorithms split the current video frame into blocks and for each block a motion vector is extracted by finding the best matching block in the previous frame. The best matching block is found using a cost function that measures the similarity between two blocks. Since the best block is usually in the vicinity of the position of the current block, but in the previous frame, the search for the best matching block is not performed in the entire frame but in an area called the search window. The dimension of the search window defines the computational cost of the algorithm but also the precision of the estimation. The best block matching algorithms use a fixed dimension for the search window and show good results [5].

In this paper we present i) a group of fast block matching algorithms for motion estimation, ii) the importance of the search windows size in the precision of the estimation, iii) the algorithms that show increase in precision, and iv) a method for selecting the search windows dimensions in order

to obtain the best estimation precision without an increase of computational cost. The fast block matching algorithms gain a significant decrease of the computational cost by selecting only a small set of blocks in the search windows. The current block, or the reference block, is compared only to these blocks and the best matching block is selected from this set. This means less comparisons, so small computational cost, but also the possibility that the best block is not found between the block in the search set. The main difference between the existing block matching algorithms is the search pattern, i.e., the method for selecting the blocks in the search set. We can classify these algorithms into two categories: fixed number of steps and variable number of steps. The ones in the first category have fixed number of steps and the estimation precision does not depend on the search window dimensions. The ones in the second category usually start with a search step, i.e., half the search window parameter, so the precision of the estimation depends on the search window size. The method proposed in this paper uses a variable size search window in order to obtain better motion estimation with no increase in the overall computational cost. The selection of the search window size is done in relation to the ratio between the frame size and the motion area size.

The rest of the paper is organized as follows. In Section II, we briefly describe the block matching motion estimation method; then, we present four of the best fast block algorithms, one with fixed number of steps and three with variable number of steps. In Section III, we present the proposed search windows' dimensions selection method. The comparative experimental results of the presented algorithms for different window sizes are shown in Section IV. Finally, the conclusions are provided in Section V.

II. BLOCK MATCHING MOTION ESTIMATION

Motion information is the most significant information of videos. A video can be regarded as a group of successive frames or images that together convey a specific message. Therefore, extracting features of videos can be accomplished using existing methods for extracting image features. There is a very important feature of the video that does not exist for images, namely the motion.

The concept of motion refers to the variation in time of the spatial position and applies to existing objects or the entire frame, in the case of camera motion. In the first case the background is not changing position and only the objects in the video have a variation of position over time. In the second case the whole frame changes over time due to the camera motion.

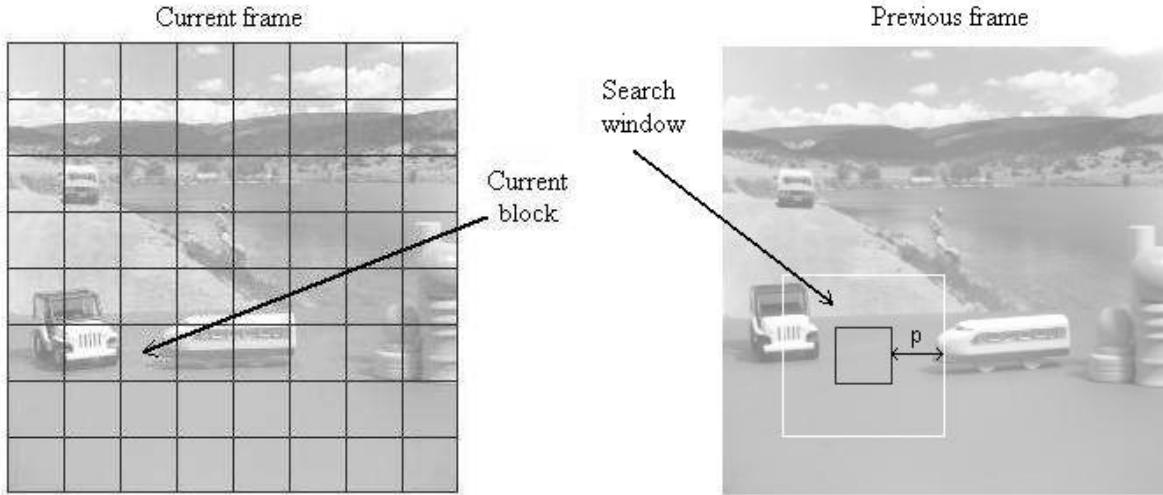


Figure 1. Schematic representation of block matching motion estimation.

There is also the case in which there is both camera motion and moving objects. Extracting video motion information is called motion estimation. This operation is done by comparing two by two successive video frames using different methods.

Motion estimation is used in the processes of video compression/decompression. In the compression phase the motion is estimated by comparing the current frame with the previous frame. Then, using the motion information and the previous frame, a motion compensated image of the current frame is build and the difference between the current frame and the motion compensated frame is computed. Also, instead of compressing the current frame, the motion information and the error frame is compressed. In this way higher rates of compression are achieved.

There are two classes of motion estimation methods:

- Motion flow estimation: for each pixel of the frame a motion vector is determined. The advantages of the methods in this class are high accuracy and high resolution of estimation. The main disadvantage is the computational complexity.
- Motion estimation based on blocks of pixels, known as block matching motion estimation. The basic idea of block matching algorithms is dividing the current frame in a matrix of non-overlapping macro blocks and determining motion vectors for each block of video frame (Figure 1).

The main advantage of the methods of the second class is that the size of pixel blocks can be chosen depending on the particular application. So for applications requiring high precision of motion vector estimation the size of the pixel blocks can be smaller and the computational complexity will increase, while for applications where speed is more important than the accuracy the blocks of pixels can be larger. If the size of the blocks is chosen 1x1 we obtain the motion flow estimation.

After the splitting of the frame into pixel blocks the motion of each block is estimated as follows: the block in the current frame is compared to all overlapping blocks in the search window. The search window is an area in the previous frame obtained by selecting the corresponding block, the block with the same spatial position as the current block, and adding pixels in each direction. The parameter is called the search window parameter. Its value determines the estimation precision and the computational complexity. Higher value implies higher chances of correct estimation and high number of blocks in the search window.

Determining the best block is made based on a cost function. For each block in the current frame a set of cost function values is determined by comparing it to all overlapping block in the search window. The block with the minimum cost is the best matching block. The most used cost functions are the Mean Absolute Difference (MAD) and the Mean Squared Error (MSE) given by (1) and (2), respectively:

$$MAD = \frac{1}{N^2} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} |C_{ij} - P_{ij}|, \quad (1)$$

$$MSE = \frac{1}{N^2} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} (C_{ij} - P_{ij})^2, \quad (2)$$

where N is the block size, C_{ij} are the pixels values of the block in the current frame, and P_{ij} are the pixels values of the block in the previous frame.

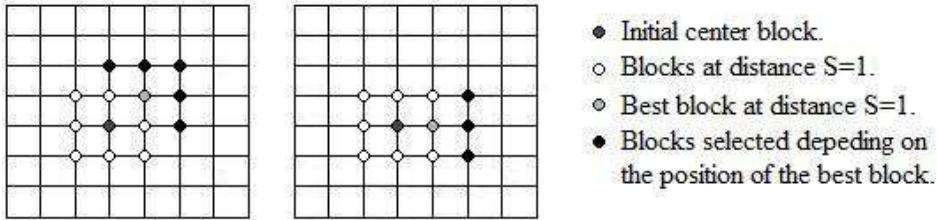


Figure 2. Example of blocks selected at every step for the New Three Step Search algorithm.

After the best matching block is determined the motion vector is computed as the difference between the spatial position of the current block and the spatial position of the best block. The resulting motion vector has two components for each direction, horizontal and vertical.

Based on the motion vectors of all blocks from the current frame and blocks in the previous frame the motion compensated frame is computed. The estimation precision or accuracy is determined using the Peak Signal to Noise Ratio (PSNR) between the current frame and the motion compensated frame, i.e.,

$$PSNR = 10 \lg \left(\frac{Vpp^2}{MSE} \right), \quad (3)$$

where Vpp is the peak to peak value of the original data and MSE is the mean square error between the original data and the motion compensated data.

A. Full Search Algorithm

The first block matching motion estimation algorithm is called the Full Search (FS) algorithm, where all the overlapping blocks in the search window are used for determining the best matching block. Although this algorithm is the best in terms of prediction quality and simplicity, it is also the most inefficient in terms of arithmetic complexity. To assess the computational complexity of the FS algorithm we needed to determine the number of blocks in the search window compared with each reference block. For example, for 16x16 blocks and a search parameter we have 225 blocks in the search window.

To reduce the computational complexity new algorithms were developed with a higher quality complexity ratio. These algorithms are called suboptimal because they offer lower prediction quality than the algorithm above and are also called fast algorithms because they have lower computational complexity. These algorithms use only a set of blocks from the search window to determine the best matching block.

There are two classes of fast algorithms:

- Search Window Independent algorithms;
- Search Window Dependent algorithms.

B. Search Window Independent Algorithms

In order to properly classify the fast block matching algorithms it is important to explain the way these algorithms function and to identify the parameters that determine the affiliation of a certain algorithm to one of the classes defined earlier.

All of the fast block matching algorithms have an initial step in which a block from the current frame is compared to the correspondent block in the previous frame and a number of blocks at a distance S from the correspondent block. The number of blocks and their position is chosen different for every algorithm. The distance is defined in terms of number of pixels.

The algorithms in the first class start with an initial distance $S = 4$ and after one step the distance is halved. The algorithms stop when the distance reaches 1.

For this category of algorithms the size of the search window is 7 pixels in each direction, meaning that if we have block of dimension $N \times N$ then the size of the search window will be $(N + 7) \times (N + 7)$. We recall that the search window parameter is denoted with p . The value $p = 7$ is chosen so that the algorithms go through all their steps without reaching a block outside the search window limits.

The window independent algorithms have two important properties:

- The maximum number of verified blocks is known.
- The precision of estimation is independent of the search window dimensions, so that if the motion has a high amplitude the increase of the search window dimensions will have no effect on the estimation.

The most efficient fast block matching algorithm in this category is the New Three Step Search (NTSS) algorithm. It has good precision of estimation and low computational complexity. It does not fall into the category of interest for the proposed method but for comparison reasons we present it in the following.

C. New Three Step Search Algorithm

The NTSS algorithm compares the block in the current frame to the center block, eight blocks at distance $S = 4$ and eight blocks at a distance $S = 1$ on the horizontal and vertical axes, in the previous frame [6].

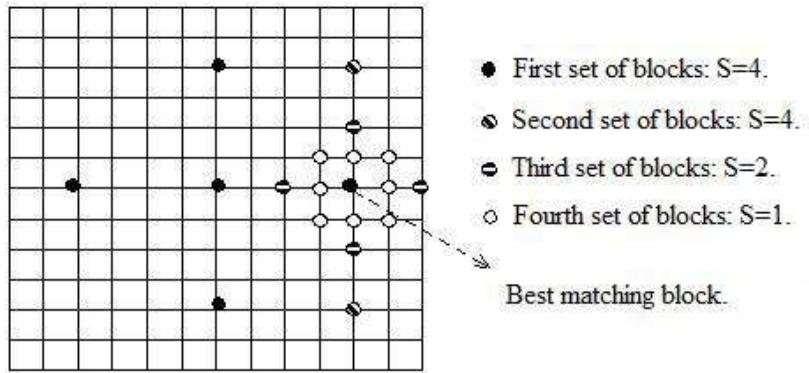


Figure 3. Example of blocks selected at every step for the Two Dimensional Logarithmic Search algorithm.

The best block from these initial 17 blocks is determined based on the cost function values. Depending on the positions of the best block we have three situations:

- 1) If the best block is the one in the centre of the search window, then the algorithm stops.
- 2) If the best block is one of the blocks located at a distance $S=1$, then its neighbors are compared with the current block and the best block is determined as the block with the minimum cost function value.
- 3) If the best block is one of the blocks located at a distance $S=4$, then the block is set as the new center, the distance is halved and all eight blocks at distance S are verified. The algorithm stops when the distance is one.

To decrease the number of blocks compared and to eliminate the re-evaluation of some blocks, the neighbors selected in the second case (when the best block is one of the blocks located at distance $S=1$) depend on the position of the best block as shown in Figure 2.

D. Search Window Dependent Algorithms

As compared to the algorithms in the first class the algorithm in this class start with an initial search distance equal to half the search window parameter p . The selection of the blocks that are used for computing the cost values depends on the algorithm.

It is obvious that compared to the first class the algorithms in the second class have a maximum number of verified blocks that depends on the search window dimensions. Also, if the search window size increases the precision of estimation will increase in the case with high amplitude motion.

In the following, we present the most important algorithms in this class.

E. Two Dimensional Logarithmic Search Algorithm

Two Dimensional Logarithmic Search (TDLS) algorithm selects at every step the center block and four blocks at a distance S on the horizontal and vertical axes. The initial distance is chosen as half the search window parameter

$p = 7$. If the search parameter is an odd number then S is chosen as the rounded value of $p/2$.

After the selection of the initial distance, the block from the current frame is compared to the center block, i.e., the corresponding block in the previous frame, and the four blocks at distance S on the horizontal and vertical axes. The comparison is done by computing the cost functions. The block that gives the lowest cost function is selected for the next step.

If the block selected at the first step is the center block then the search distance S is halved, else the selected block is set as the new center and the first step is repeated.

When the search distance becomes equal to one the center block and all its neighbors are compared to the block in the current frame and the best matching block is selected as the block with the minimum cost function value.

F. Orthogonal Search Algorithm

The Orthogonal Search (OS) algorithm is a combination of the TDLS algorithm and the Three Step Search (TSS) algorithm. The TSS algorithm is the first fast block matching algorithm and is independent of the search window dimensions, so it belongs to the first class of algorithms. The similarity between OS and TSS algorithms is the number of steps.

The initial search distance is chosen as half the search window parameter. The OS algorithm has the following 3 steps:

- 1) The block from the current frame is compared to the center block and the two blocks at distance S on the horizontal axis. The block with the minimum cost function value is set as the new center.
- 2) The center block and the two blocks at distance S on the vertical axis are verified, and the new center is selected as the block with the lowest cost.
- 3) If the distance parameter S is bigger than one then the distance is halved and steps 1 and 2 are repeated. Else, the last center block is the best matching block.

G. Adaptive Rood Pattern Search Algorithm

The Adaptive Rood Pattern Search (ARPS) algorithm uses the motion information of the neighboring block in the left. This is helpful if the current block and its neighbor on the left belong to the same object in the frame; in this case, their motion is similar [7]. The steps of the ARPS algorithm are:

- 1) The block from the current frame is compared to the center block, four blocks at distance S on the horizontal and vertical axes, and the block indicated by the motion vector of the neighbor block in the left. The initial search distance S is selected as the maximum between the absolute values of the neighboring motion vector.

- 2) The block with the minimum cost function value is set as the new center. The search distance is set to 1 and the centre block together with its four axis neighbors are evaluated.

- 3) If the block with the minimum cost is in the centre, then the algorithm stops; consequently, this is the best block. Else, step 2 is repeated.

III. PROPOSED SEARCH WINDOW SELECTION

The proposed method is based on the simulation results that showed, as the theory stated, that by increasing the size of the search window the precision of estimation increases. The increase of the search window dimensions has an unwanted effect, i.e., an increase of the number of verified blocks.

To highlight these observations we present in Table I the PSNR between the current frame and the motion compensated frame for all, of the above presented, fast block matching algorithms. Also, in Table II we present the total number of blocks verified. The simulations were done for different search window dimensions, 8x8 pixel blocks and the computer generated video sequence "Motion."

From Table I it can be observed that for the search window dependent algorithms the estimation precision increases with the increase of the search window.

In Table II it can be seen that along with the increase of the PSNR there is also an increase of the number of verified blocks.

Based on the results presented in both tables we make two observations:

- 1) First of all, for the search window dependent algorithms the estimation precision parameter for larger search windows exceeds both the FS algorithm and the NTSS.

- 2) Second, although for all the search window dependent algorithms the number of verified blocks increases, the computational complexity remains significantly smaller than for the FS algorithms, and in some cases, like ARPS, even smaller than for the NTSS algorithm.

TABLE I. PSNR VALUES FOR DIFFERENT WINDOW SIZES

Algorithm	Search Window Parameter			
	p=7	p=15	p=31	p=63
FS	30.78	-	-	-
NTSS	30.49	30.49	30.49	30.49
TDLS	29.3	32.49	33.8	34.69
OS	28.88	31.54	32.18	33.45
ARPS	29.47	31.8	32.73	33.21

TABLE II. OVERALL NUMBER OF VERIFIED BLOCKS FOR DIFFERENT WINDOW SIZES

Algorithm	Search Window Parameter			
	p=7	p=15	p=31	p=63
FS	921600	-	-	-
NTSS	79898	79898	79898	79898
TDLS	74349	93676	111607	128330
OS	53248	69632	85988	102300
ARPS	28548	30038	30462	30642

There is also one disadvantage in increasing the search window. For sequences with low motion amplitude there is small or even zero increase in the estimation precision but the increase of the number of verified blocks remains. So the primary concern is determining a way of obtaining the best estimation precision with the lowest computational cost.

The goal of proposed method is the optimum search window size selection. This is done in three simple steps:

- 1) Motion area detection.
- 2) Search window parameter computation.
- 3) Motion estimation.

The detection of the area where motion exists is done by simple difference between the current frame and the previous frame and two morphological operations to eliminate the misdetection of motion due to variations of pixel intensity.

The search window parameter is computed based on the ratio between the entire frame and the motion area, as:

$$p = 2^{r+1} - 1, \quad (4)$$

$$r = \left\lceil \frac{A_x \times A_y}{F_x \times F_y} \right\rceil, \quad (5)$$

where $\lceil \cdot \rceil$ is the round operator, A_x and A_y are the motion area dimension, F_x and F_y are the frame dimensions, and p is the search window parameter. The parameter p is

chosen as a power of 2, minus one, so that when computing the search parameter S (as half the search window dimension) it will be also a power of 2.

By selecting the search window size this way we obtain the best PSNR with the lowest computational complexity. This means that if the ratio r is high the PSNR will be higher without increasing too much the computational complexity. If the ratio is close to one the window size will be low, with $p = 7$, the usual value for the window size independent fast block matching algorithms.

Also by applying the motion estimation algorithms only to the area where motion exists there will be a significant decrease of the computational complexity with small loss of estimation precision.

IV. SIMULATION RESULTS

In this section we present the comparative results of the presented fast block matching algorithms for fixed search window size and variable search window size selected with the proposed method.

The fast block matching algorithms presented in section II were implemented using Matlab and we used a set of video sequences containing monochrome videos and color videos of different sizes, some of the videos artificially generated and some from the real life. All the videos were obtained from test sequences databases commonly used for motion estimation.

We present the results for the test sequence "Motion:" a computer generated monochrome sequence with 10 frames, 512x512 pixels and high amplitude motion.

In concordance with step 3 of the proposed search window selection method presented in Section III, in this section the comparative results are split into two distinct scenarios: one to compare the results for the estimation when the improved method is applied to the entire frame, i.e., Variable Search Window – Full Frame (VSW-FF), and one to compare the results for the estimation when the improved method is applied only to the area where motion is detected, i.e., Variable Search Window – Motion Area (VSW-MA). The results for the two scenarios are compared to the results of the original segmentation method with fixed search window dimensions, i.e., Fixed Search Window (FSW).

For the first scenario, after the detection of the area with motion and the selection of the search window dimension, the algorithms are applied only to the area where motion is present. In this case, if the ratio between the entire frame and the area with motion is one then the PSNR will be slightly smaller than the one obtained with the initial motion estimation method but the overall number of blocks verified decreases. If the search window is bigger then the PSNR increases if the motion has high amplitude and or decreases if the motion has low amplitude. The decrease of the PSNR is due to the small intensity differences of the pixels from the areas of the frame where the algorithms are not applied.

Also, along with the decrease of the PSNR we obtain a decrease of the overall number of blocks verified. For this reason it is important to see the comparative result of the PSNR and Nb ratio.

For the second scenario the modified motion estimation method uses the selection of the search window dimension by detecting the area where motion is present. If the ratio between the entire frame and the area with motion is high then the search window size is bigger, according to equation (4). In this case, the precision of estimation will increase and also the overall number of verified blocks will increase.

To compare the results we present the PSNR between the current frame and the motion compensated frame of the initial motion estimation method using the FS, NTSS, TDLS, OS, and ARPS block matching algorithms, and of the proposed estimation method with variable search window size. We also present the overall number of blocks verified all the algorithms in the two situations.

For the first scenario, VSW-MA, and the computer generated sequence "Motion" we observe the following:

- First of all, as expected the NTSS algorithm results are independent of the search window size so the PSNR between the current frame and the motion compensated frame decreases for the proposed method. This happens because the algorithm is applied only to the motion area and due to small changes in the frames that are not determined by motion. The overall number of blocks verified decreases for the same reason above.
- For the other three algorithms if we compare the results presented in Figure 4, for the fixed size search window and for the variable size search window and motion area, we observe that there are two situations depending of the existence of camera motion or the occlusion of objects.
- In the first case, for frames 2-7 and 9, there is no camera motion and no occlusion. We see from Figure 4a-c that the PSNR between the current frame and the motion compensated frame obtained from the previous frame and the motion vectors increases for all of the search window dependent algorithms, TDLS, OS, and ARPS.
- From Figure 4d we observe that the highest increase in PSNR is obtained for the TDLS algorithm and that the precision of estimation for TDLS and ARPS exceeds the results of the FS algorithm. For the OS algorithm, although there is an increase of the PSNR, for some frames the precision of the motion estimation is lower than the one of the FS algorithm. In terms of PSNR between the current frame and the motion compensated frame we can conclude that the best results are obtained by the TDLS algorithm.
- Regarding the computational complexity, evaluated through the overall number of blocks verified, we observe from Figure 5a that for frames 1-7 and 9 there is a decrease in computational complexity because all the algorithms are applied only to the area where the motion is detected. For a better observation of the results regarding the computational complexity we have not represented the results for the FS algorithm, results that are constant for all the frames and are equal to 10^5 .

- In Figure 5b is represented the computational complexity for the three search window dependent algorithms. We can observe that the TDLS algorithm, that has the highest PSNR, has also the highest number of verified blocks. The OS and ARPS algorithms have lower computational complexity. In terms of the overall number of verified blocks we can conclude that both OS and ARPS algorithms have good results.
- So in the first discussed case, for the frames without camera motion, we can conclude that all of the three algorithms show good results with increase of the PSNR and a decrease of the overall number of verified blocks and that the windows parameter selection method has good results.
- In the second case, for frame 1, the existence of occlusion leads to smaller PSNR even if the search window increases. For frame 8 the existence of camera motion leads to the detection of a motion area almost equal to the entire frame. In this case the ratio between the motion area and the entire frame is close to 1 and the search window parameter is set to $p = 7$. In this case there is no increase in the PSNR for none of the algorithms and a small decrease in computational complexity due to the fact that the detected motion area is smaller than the entire frame.
- In this case the TDLS and ARPS algorithms have similar results in terms of PSNR but the ARPS algorithm has lower computational complexity.

As a conclusion for this scenario of the proposed motion estimation method we can state that:

- The computational complexity, evaluated through the overall number of blocks verified, decreases.
- The estimation precision, evaluated through the PSNR, increases in case of large amplitude motion or decreases slightly in case of small amplitude motion.

In the second scenario, i.e., VSW-FF, the results are presented in Figures 6 and 7. Based on these results we make the following observations:

- The NTSS algorithm has the same results for the proposed method as the initial method both in terms of PSNR and number of blocks verified. This was expected because the NTSS algorithm is independent of the search window.
- All of the three search window dependent algorithms show an increase in PSNR, Figure 6a-c, and also an increase in the overall number of verified blocks, Figure 7a. From the same figures we observe that for the proposed method in the case of the TDLS and ARPS algorithms the PSNR is higher than the PSNR for the FS algorithm. Also, even if the number of verified blocks increases is significant smaller than the number of verified blocks for the FS algorithm.
- From Figure 6d we observe that in terms of PSNR between the current frame and the motion compensated frame the algorithm with the best

results is the TDLS algorithm, followed by ARPS and OS.

- From Figure 7b it can be seen that in terms of the overall number of verified blocks the ARPS algorithm has the best results, followed by OS and TDLS.

In conclusion, for the variable search window parameter method applied to the entire frame, the results show that all of the algorithms have an increase of PSNR, when there is no camera motion and the motion area is at least two times smaller than the entire frame. If there is camera motion then the value for the search window parameter will be equal to the value for the original method and the results the same.

For applications that require high precision of estimation and no constraints in computational complexity the proposed method can be applied to the entire frame. For applications that require low computational complexity the method can be applied only to the area where motion is detected with little loss or significant gain in PSNR.

In Tables III and IV, we represented the mean value of the PSNR and the total number of verified blocks, for the video sequence "Motion," for the initial motion estimation method with fixed search window parameter and the two variations of the proposed variable search window method.

TABLE III. MEAN PSNR VALUES FOR SEQUENCE "MOTION"

Algorithm	Search Window Dimension Type		
	FSW	VSW - MA	VSW - FF
FS	31.34	-	-
NTSS	31.06	30.67	31.06
TDLS	30.05	32.59	32.69
OS	29.74	31.24	31.39
ARPS	30.32	31.84	31.96

TABLE IV. OVERALL NUMBER OF VERIFIED BLOCKS FOR SEQUENCE "MOTION"

Algorithm	Search Window Dimension Type		
	FSW	VSW - MA	VSW - FF
FS	921600	-	-
NTSS	87817	27399	87817
TDLS	79923	31154	100573
OS	53248	19363	69629
ARPS	34547	16851	35256

The second set of results is for the video sequence "Hall Monitor," a real life color video sequence with 144x176 pixels. In Tables V and VI, we presented the results for the original motion estimation method with fixed search window and the two variations of the proposed

method. Based on these results we can make the following observations:

- In the first case, when the variable search window method is applied to the area where motion is detected, the PSNR between the current frame and the motion compensated frame decreases for all the presented algorithms but also the overall number of verified blocks decreases.
- In the second case, when the variable search window method is applied to the entire frame, the PSNR also decreases for all of the algorithm and the overall number of verified blocks increases.
- The results can be explained by the existence of low amplitude motion, this means that a good precision can be obtained using a small search window, and by the existence of illumination variations, which lead to an increase of the MSE and a decrease of PSNR.

TABLE V. MEAN PSNR VALUES FOR SEQUENCE "HALL MONITOR"

Algorithm	Search Window Dimension Type		
	FSW	VSW - MA	VSW - FF
FS	31.01	-	-
NTSS	30.82	29.89	30.82
TDLS	30.25	29.35	30.11
OS	29.88	28.99	29.66
ARPS	29.85	28.01	28.64

TABLE VI. OVEROALL NUMBER OF VERIFIED BLOCKS FOR SEQUENCE "HALL MONITOR"

Algorithm	Search Window Dimension Type		
	FSW	VSW - MA	VSW - FF
FS	89100	-	-
NTSS	8403	2804	8403
TDLS	7424	3031	9642
OS	5148	2177	7084
ARPS	2956	951	2628

V. CONCLUSIONS AND FUTURE WORK

In this paper we have evaluated the importance of the search window dimensions for fast block matching algorithms for motion estimation and a method for selecting the search window parameter depending of the area where motion is detected.

By evaluating twelve fast block matching algorithms for motion estimation, with different block sizes and search window dimensions, we concluded in [1] that the existing fast block matching algorithms can be split into two categories: fixed number of steps and thus independent of the

search window dimensions, and variable number of steps and thus dependent on the search window dimensions. The results presented in [1] showed that for increased search windows some of the algorithms in the second category show an increase of the PSNR between the current frame and the motion compensated frame obtained from the previous frame and the estimated motion vectors.

We presented four fast block matching algorithms for motion estimation, one with fixed number of steps, and thus independent of the search window dimensions, and three with variable number of steps that depend of the search window dimensions. As shown in Tables I and II for the three algorithms with variable number of steps by increasing the search window parameter we obtain an increase of the motion estimation precision but also an increase of the computational complexity.

The basic idea behind the method proposed in this paper is to select the search window dimensions in such a manner that lead to good precision of estimation and low computational complexity.

The proposed motion estimation method uses a variable search window dimension depending on the detection of the area where motion is present. The detection of motion is done by simple differencing between the current frame and the previous frame and two morphological operations. The search window parameter, that defines how many pixels the search window is extended around the current block, is computed according to the ratio between the size of the entire frame and the size of the area with motion, as shown in equations 4 and 5.

We have evaluated the proposed method in two cases: when the algorithms are applied only to the area where motion is present and when the algorithms are applied to the entire frame. The first set of simulation results were obtained for a computer generated video sequence with high amplitude motion.

In the first case we observed that, although for some frames a small decrease of the PSNR may occur, the mean PSNR for the entire sequence increases and the overall number of verified blocks decrease significantly. The simulation results show that some algorithms have better results in estimation precisions, the TDLS algorithm, while others show a more significant decrease in computational complexity, the ARPS algorithm. Depending of the application we can use one or another.

In the second case we observed that the PSNR increases for all the frames, when the search window parameter increases, but also the overall number of verified blocks increases. Similar to the result in the first case, the TDLS algorithm shows higher increase in PSNR compared to the OS and the ARPS algorithms, and the ARPS algorithm shows lower increase of computational complexity compared to the TDLS and the OS algorithms. A very important observation is that all of the three window size dependent algorithms obtain better precision of estimation than the first block matching motion estimation algorithm, the FS algorithm, and with significant lower computational complexity.

The algorithms using the proposed search window selection method we were also applied to real life video test sequences. The simulation for these test sequences also included the two variations described above.

From this case of our simulation results we drew the following conclusion:

- The search window parameter chosen by detection of the motion area leads to two situation.
- First, when the sequences contain camera motion of significant illumination variations, the value of the search parameter is low and equal to the value recommended for the fast block matching algorithms. In this case there the results are the ones from which we started.
- Second, when there is no camera motion, the illumination variations are low enough and the objects in motion occupy an area much smaller than the entire frame, the search window parameter value is higher than the one in the first case.
- In this case, for the first scenario, when the algorithms are applied to the entire frame, the results for the real life test sequences show a small decrease in estimation precision and an increase in computational complexity. This is explainable by the existence of low amplitude motion that means no increase in estimation precision when the search window parameter increases, by the fact that in the areas without motion there are changes in pixel values due to illumination, changes that can be compensated by motion estimation with a low search parameter, and also by the search pattern used by the algorithms. The computational complexity increases because in the areas without motion many blocks are verified even if is not necessary.
- For the second scenario, when the algorithms are applied only to the motion area, the estimation precision decreases slightly but the computational complexity decreases significantly. The decrease in estimation precision is explainable by the illumination variations in the areas not used, variations that can be compensated by motion estimation with a low search window parameter. The decrease in computational complexity is high and it is a very important aspect that can be exploited.

- As an overall conclusion of the presented method we can definitely say that the proposed search window parameter method show good results in estimation precision when the test sequences contain objects with high amplitude motion and also good results in computational complexity for the second scenario presented.

For future work we consider the idea of using the motion area detection for selecting the search window parameter value, by applying the algorithms with the selected parameter value only to the motion area and by applying the algorithms with the lowest parameter value for the areas without motion. Also we consider evaluating the results for the proposed method and the presented algorithms for different block dimensions.

ACKNOWLEDGMENT

This work was supported by the UEFISCDI Romania under the Grant PN-II-RU-TE no. 7/05.08.2010.

REFERENCES

- [1] I. Pîrnog, C. Anghel, A. A. Enescu, and C. Paleologu, "Evaluation of Fast Algorithms for Motion Estimation," AICT 2011, The Seventh Advanced International Conference on Telecommunications, pp. 107-111, Mar. 2011.
- [2] Z. Chen, "Efficient Block Matching Algorithm for Motion Estimation," International Journal of Signal Processing, 5(2):133-137, 2009.
- [3] ISO/MPEG N4358, "Text of ISO/IEC Final Draft International Standard 15938-3 Information Technology - Multimedia Content Description Interface - Part 3 Visual," MPEG Video Group, Sydney, July 2001.
- [4] A. Barjatya, "Block Matching Algorithms For Motion Estimation," Final Project Paper, 2004.
- [5] Y. C. Lin and S. C. Tai, "Fast Full-Search Block-Matching Algorithm for Motion-Compensated Video Compression," IEEE Transactions on Communications, 45(5):527-531, 1997.
- [6] R. Li, B. Zeng, and M. L. Liou, "A New Three-Step Search Algorithm for Block Motion Estimation," IEEE Trans. Circuits and Systems for Video Technology, vol 4., no. 4, pp. 438-442, Aug. 1994.
- [7] S. Jamkar, S. Belhe, S. Dravid, and M. S. Sutaone, "A comparison of block-matching search algorithms in motion estimation," Proceedings of the 15th International Conference on Computer Communication, pp. 730 – 739, Mumbai, India, 2002.

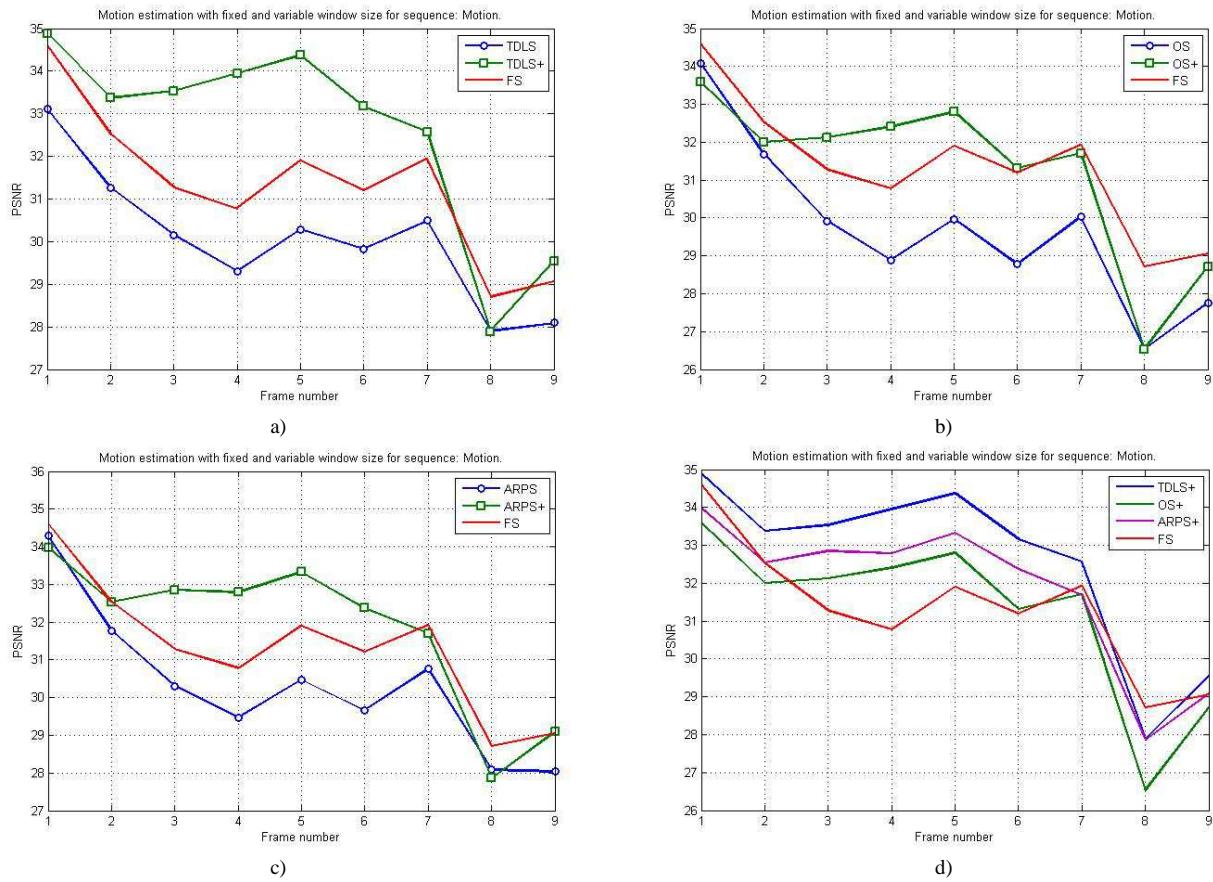


Figure 4. Comparative simulation results for video sequence "Motion" for 8x8 pixel blocks with fixed and variable search window size and motion area. a) Two-Dimensional Logarithmic Search. b) Orthogonal Search. c) Adaptive Rood Pattern Search. d) Comparative results of the three algorithms with variable search window size.

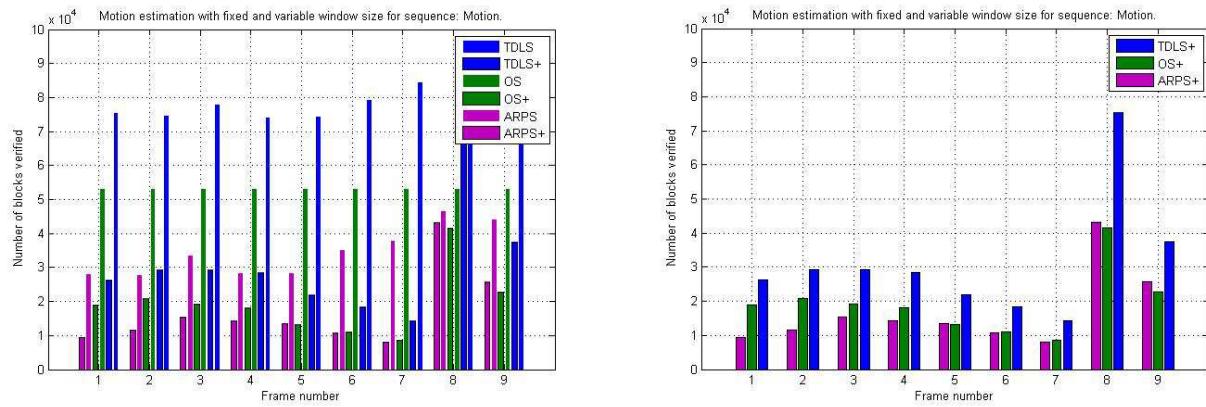


Figure 5. Overall number of verified blocks for block matching motion estimation for the video sequence "Motion" with 8x8 pixel blocks. a) Comparative results for fixed and variable search window size. b) Comparative results for the three algorithms in the case of variable search window size.

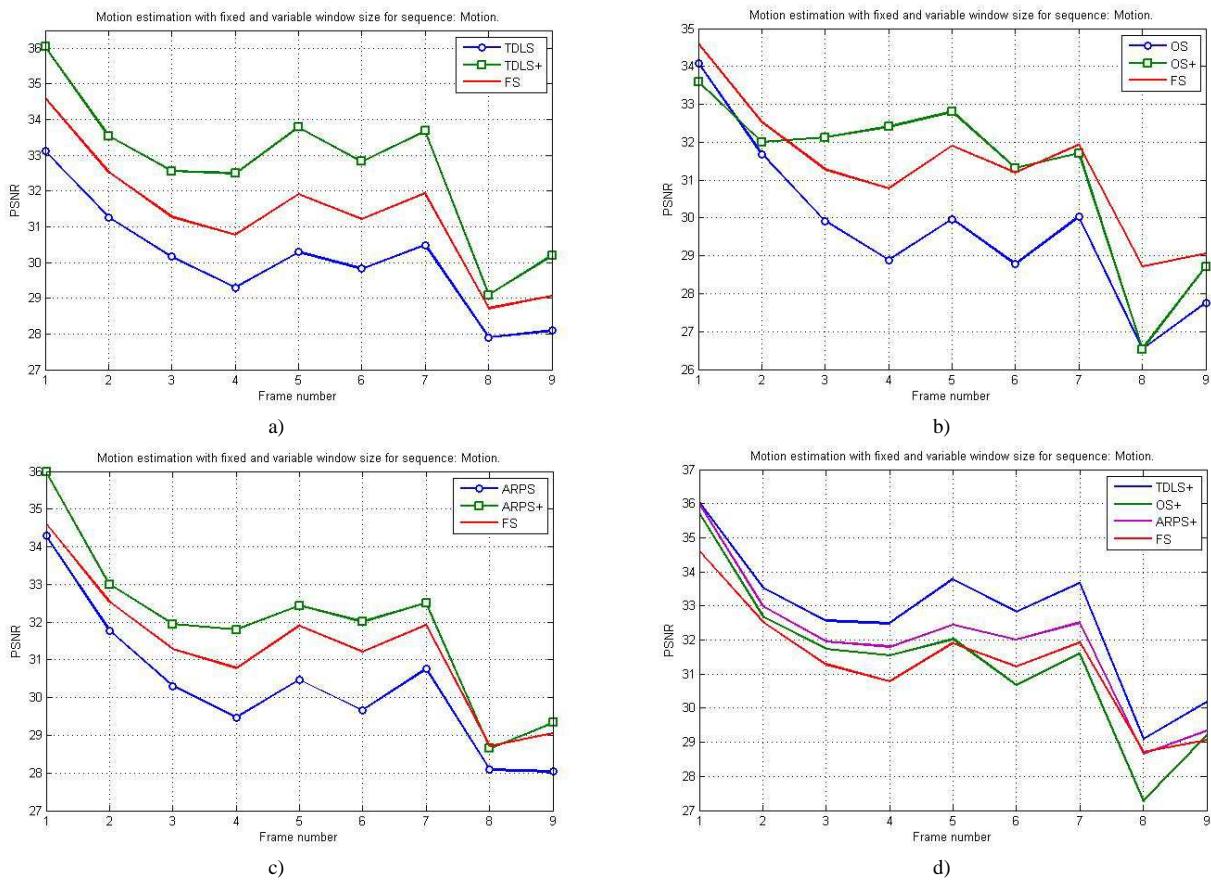


Figure 6. Comparative simulation results for video sequence "Motion" for 8x8 pixel blocks with fixed and variable search window size and full frame. a) Two-Dimensional Logarithmic Search. b) Orthogonal Search. c) Adaptive Rood Pattern Search. d) Comparative results of the three algorithms with variable search window size.

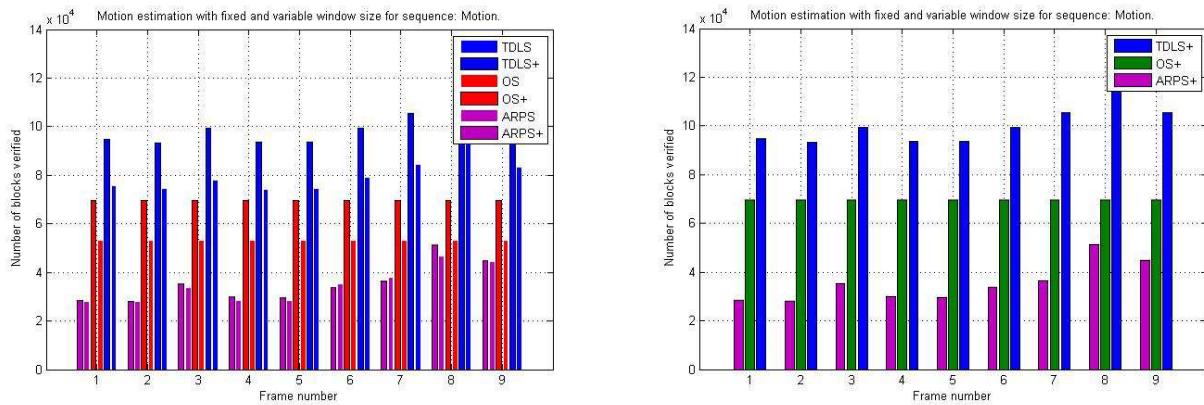


Figure 7. Overall number of verified blocks for block matching motion estimation for the video sequence "Motion" with 8x8 pixel blocks and full frame. a) Comparative results for fixed and variable search window size. b) Comparative results for the three algorithms in the case of variable search window size.

Automated Categorisation of E-Journals by Synonym Analysis of n -grams

Richard Hussey, Shirley Williams, Richard Mitchell

School of Systems Engineering

University of Reading

Reading, United Kingdom

{r.j.hussey, shirley.williams, r.j.mitchell}@reading.ac.uk

Abstract—Automatic keyword or keyphrase extraction is concerned with assigning keyphrases to documents based on words from within the document. Previous studies have shown that in a significant number of cases author-supplied keywords are not appropriate for the document to which they are attached. This can either be because they represent what the author *believes* a paper is about not what it actually is, or because they include keyphrases which are more classificatory than explanatory e.g., “University of Poppleton” instead of “Knowledge Discovery in Databases”. Thus, there is a need for a system that can generate an appropriate and diverse range of keyphrases that reflect the document. This paper proposes two possible solutions that examine the synonyms of words and phrases in the document to find the underlying themes, and presents these as appropriate keyphrases. Using three different freely available thesauri, the work undertaken examines two different methods of producing keywords and compares the outcomes across multiple strands in the timeline. The primary method explores taking n -grams of the source document phrases, and examining the synonyms of these, while the secondary considers grouping outputs by their synonyms. The experiments undertaken show the primary method produces good results and that the secondary method produces both good results and potential for future work. In addition, the different qualities of the thesauri are examined and it is concluded that the more entries in a thesaurus, the better it is likely to perform. The age of the thesaurus or the size of each entry does not correlate to performance.

Keywords- Automatic Tagging; Document Classification; Keyphrases; Keyword Extraction; Single Document; Synonyms; Thesaurus

I. INTRODUCTION

Keywords are words used to identify a topic, theme, or subject of a document, or to classify a document. They are used by authors of academic papers to outline the topics of the paper (such as papers about “metaphor” or “leadership”), by libraries to allow people to locate books (such as all books on “Stalin” or “romance”), and other similar uses. The keywords for a document indicate the major areas of interest within it.

A keyphrase is typically a short phrase of one to five words, which fulfils a similar purpose, but with broader scope for encapsulating a concept. While it may be considered the authors' contention, it is inferred that a short

phrase of a few linked words contains more meaning than a single word alone, e.g., the phrase “natural language processing” is more useful than just the word “language”.

Previous work by Hussey et al. [1] showed that using a thesaurus to group similar words into keyphrases produced useful results. The experiments run used the 1911 edition of *Roget's Thesaurus* [2] as the basis of the work. This paper sets out to expand upon that work by examining the results in relation to results generated by chance and, by using a number of different thesauri, to generate the keyphrase groupings, to compare the results of the different systems, and the different thesauri.

Frank et al. [3] discuss two different ways of approaching the problem of linking keyphrases to a document. The first, keyphrase assignment, uses a fixed list of keyphrases and attempts to select keyphrases that match the themes of the document. The computational problem for this approach is then to determine a mapping between documents and keyphrases using already classified documents as learning aids. The second approach, keyphrase extraction, assumes there is no restricted list and instead attempts to use phrases from the document (or ones constructed via a reference document).

Previous research [4][5] has shown that for any given group of documents with keyphrases, there are a small number which are frequently used (examples include “shopping” or “politics” [5]) and a large number with low frequency (examples include “insomnia due to quail wailing” or “streetball china” [5]). The latter set is too idiosyncratic for widespread use; generally, even reuse by the same author is unlikely. Therefore, part of the issue of both keyphrase assignment and extraction is locating the small number of useful keyphrases to apply to the documents.

The work described here is concerned with keyphrase extraction and, as such, this paper covers the background research into keyword/keyphrase generation, outlines a proposed solution to the problem, and compares the performance of manually assigning keyphrases. The main aim is to take an arbitrary document (in isolation from a corpus) and analyse the synonyms of word-level n -grams to extract automatically a set of useful and valid keywords, which reflect the themes of that document. The words of the document are analysed as a series of n -grams, which are compared to entries in a thesaurus to find their synonyms and these are ranked by frequency to determine the candidate

keywords. The secondary aim is to look at a method of grouping the theme outputs into clusters, so that the results do not just show the most common theme swamping out any others.

The rest of the paper comprises the background and state-of-the-art (Section II), the implementation (Section III) and results gained (Section IV), a discussion (Section V), and conclusions and suggestions for future work (Section VI).

II. BACKGROUND

A review of literature in the area of automatic keyword generation has shown that existing work in these areas focuses on either cross analysing a corpus of multiple documents for conclusions or extrapolating training data from manual summaries for test documents.

While manual summaries generally require multiple documents to train upon, they do not need to compare each component of the corpus to all other components. Instead, they try to extrapolate the patterns between the pairs of documents and manual summaries in the training set.

The following two sections look at firstly the manual summaries and single document approaches, and then the multiple document methods.

A. Single Documents

Single document approaches make use of manual summaries or keyphrases to achieve their results. Tuning via manual summaries attempts to replicate the process by which a human can identify the themes of a document and reduce the text down to a summary/selection of keyphrases. The general approach taken involves a collection of documents (with associated human summaries) and a given method is applied to draw relationships between the document and the summary. From this, new documents (generally a test corpus that also contains human summaries) are subject to the derived relationships to see if the summaries produced by the system are useful and usable.

For creating summaries, Goldstein et al. [6] set out a system based upon assessing every sentence of the document and calculating a ranking for its inclusion in a summary. They made use of corpora of documents for which assessor-ranked summary sentences already existed, and attempted to train the system using weighted scores for linguistic and statistical features to produce similar or identical sentences.

When getting emails I get a notice that an email has been received but when I try to view the message it is blank. I have also tried to run the repair program off the install disk but that it did not take care of the problem.

(a)
Receive emails; some emails have no subject and message body
(b)

Figure 1. An example of a) a text and b) its summary [7]

A different approach is taken by the *Stochastic Keyword Generator* [7], a proposed system for classifying help desk problems with short summaries (see Figure 1). Submitted e-mails varied in their description of the problem and often contained duplicated or redundant data. Therefore, their system attempts to create a summary similar to those manually created by the help desk staff: concise, precise, consistent, and with uniform expressions. It uses a corpus of e-mails with manual summaries, and ranks source words for inclusion based on the probability that they will occur based on the probability from its training data. This allows for words that are not explicitly in the text to appear in the summary (see Figure 2).

For producing keyphrases, Barker and Cornacchia [8] propose a system that takes into account not only the frequency of a “noun phrase” but also the head noun. For example, tracking “the Canadian Space Agency” should also track counts of “the Space Agency” or “the Agency”.

Wermter and Hahn [9] examine a method of ranking candidate keyphrases using the limited paradigmatic modifiability (LPM) of each phrase as a guide to locating phrases with low frequency but high interest to the document. This works on the principle that a given multi-word term is a number of slots that can be filled with others words instead. For example, “t cell response” contains three slots that are filled, respectively, by “t”, “cell”, and “response”. Another phrase that could fit might be “white cell response” or “the emergency response”. The probability there are no phrases that could fill the gaps (for any given combination of the original words and gaps) determines how

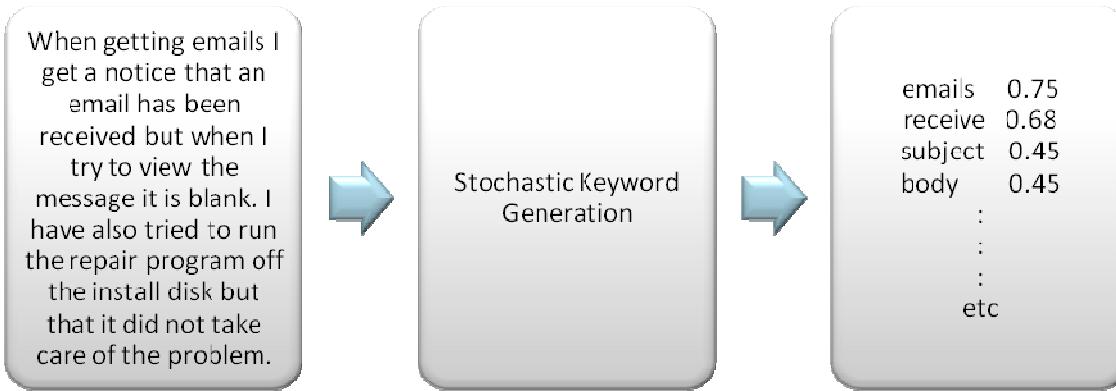


Figure 2. An example of SKG [7]

important the original phrase is, regardless of its actual frequency.

B. Multiple Documents

Multiple document approaches take a corpus and attempt to analyse relationships between the component elements to create methods for dealing with unseen elements. Most of these approaches are based on examining parts of an individual document in the corpus and then examining how that differs across the other documents.

“*TagAssist*” [4] makes use of a continually updated corpus of blog posts (supplied by [5]) and author-supplied tags to suggest tags for new blog posts. The system compares the author’s tags and content of blog posts to work out the relationships that prompt the former to be chosen to represent the latter. Their baseline system works on a simple frequency count for determining output. Evaluated by ten human judges (unaware of which system produced each tags), the results showed that the original tags were the most appropriate (48.85%) with *TagAssist* coming in second (42.10%), and the baseline system last (30.05%).

The *C-Value* [10] is presented as a method for ranking “term words”, taking into account phrase length and frequency of its occurrence as a sub-string of another phrase. It makes use of a linguistic filter, expressed as a regular expression, to ensure that only particular strings can be considered as candidate terms. Three filters were tested:

- Filter 1 – Noun ⁺ Noun
- Filter 2 – (Adjective | Noun) ⁺ Noun
- Filter 3 – ((Adjective | Noun) ⁺ | ((Adjective | Noun) ^{*} (Noun Preposition) [?]) (Adjective | Noun) ^{*}) Noun

The more permissive filters, which accepted more grammatical structures, were found to perform more poorly, though all filters performed better than the baseline.

The *C-Value* is extended by the *NC-Value* [10], which adds a context weight to the calculation to determine which words surrounding the term are important.

The *SNC-Value* [11] (or *TRUCKS*) extends the *NC-Value* work, combining it with [12], to use contextual information surrounding the text to improve further the weightings used in the *NC-Value*.

Extra data may be used to gain more information on the relationships between the components, often gained from reference documents. Joshi and Motwani [13] make use of a thesaurus to obtain extra meaning from keywords. Their program, “*TermsNet*”, can observe keywords in their original context in attempt to link keywords through a framework of linked terms, with directional relevance. This allows them to discover the “non-obvious” but related terms. For example, the term ‘eurail’ strongly suggests ‘Europe’ and ‘railways’, but neither suggest ‘eurail’ with the same strength. This means that ‘eurail’ is a non-obvious but highly relevant search keyword for both ‘Europe’ and ‘railway’.

Scott and Matwin [14] use the *WordNet* lexical database [15] to find the hyponyms and feed this information to the Ripper machine learning system. The authors tested it against the DigiTrad folk song database [16], the Reuters-21578 news corpus [17], and a selection of USENET articles. They concluded that the system works better on

documents written with “extended or unusual vocabulary” or which were authored collaboratively between several people.

Wei et al. [18] demonstrate such a system that uses *WordNet* to generate keywords for song lyrics. Their approach clusters the words of a song using *WordNet*’s data to link words across the song. Keywords are then found at the centres of these links.

C. Background Conclusions

In conclusion, the literature review determined that work such as [13] or [14] used similar methods to the ones outlined in this paper. However, there are some key differences.

Joshi and Motwani [13] used a system of weighted links, which can differ in value from one side to another (in some cases being uni-directional as the weight ‘removes’ the link by setting it to a value of zero). This would differ from the proposed system, as the thesaurus does not contain the lexical knowledge to weight the links and a link from one synonym group to another is reciprocated in kind.

In [14], hyponyms were used, rather than synonyms. Hyponyms are words or phrases that share a *type-of* relationship, e.g. scarlet and vermillion are hyponyms of red, which is in turn a hyponym of colour. The proposed system would instead use synonyms: different words with almost identical or similar meanings.

III. IMPLEMENTATION

The basis of the work presented here is the examination of a document with reference to its synonyms and therefore the main bulk of the coding of the system related to this and the associated thesaurus file. Three input thesauri were used for analysis of the corpora, and these were Roget’s “*Thesaurus of English Words and Phrases*” [16], Miller’s “*WordNet*” [14], and Grady Ward’s “*Moby Thesaurus*” [19].

The system was tested on a number of papers taken from a collection of online e-journals, Academics Conferences International (ACI) [20]. There were five e-journals in this collection, each on a different topic, and they were analysed separately. The topics were *Business Research Methods* (EJBRM), *E-Government* (EJEG), *E-Learning* (EJEL), *Information Systems Evaluation* (EJSE), and *Knowledge Management* (EJKM).

For each of the methods described below the thesaurus was loaded into the program and stored as a list of linked pairs of data, consisting of a unique Key (base word in the thesaurus) and an associated Value (its synonyms). The keys and values ranged from unigram word entries up to 7-gram phrases.

The project was split into a number of studies, and all the results were compared to a set of results generated by chance. The studies undertaken were the chance study, the unigram system, the *n*-gram study, and the clustering study. The following sections outline these approaches. The results are presented in Section IV.

A. Chance Study

For the chance study, the words from the source document were split into a list of individual words. From this list, a start point was chosen at random and a number of contiguous words were strung together to form a keyphrase. After each word was added, there was a chance that no further words would be added and this chance increased after each word so that it was more likely to produce shorter keyphrases than longer. The maximum length of the keyphrase was set at $n = 7$. The algorithm used was:

- Randomly select a word in the source document to act as a starting point.
- After each word is added, generate a random number less than or equal to n . If this number is greater than the number of words already in the phrase, add another word.
- Repeat until r keyphrases have been produced (in this study, r was chosen to be 5).

This algorithm is shown in Figure 3.

B. Unigram System

The Unigram system was designed to act as a baseline for the experiments. The source text was split into a list of unigrams, and a count of the number of times each appeared in the source document occurred. The unigrams were then stemmed (to remove plurals, derivations, etc.) using the Porter Stemming Algorithm [21], and added to the list with combined frequencies from each of the unigrams that reduced to that stem. The resultant corpus of unigrams and stems was then compared to the entries in the thesaurus. Only the highest frequency keyword was output from the unigram system.

- For each n -gram in the thesaurus, compare the n -gram to the associated synonyms.
- For each synonym that matches, add the word to a list, and increase its frequency value by the value of the n -gram.
- Sort the list by frequency and output the top r ranked items (in this study, r was chosen to be 1).

C. The n -gram study

Following the results of the unigram study, the experiment was extended to examine the effects of multi-gram words on the output of the system. This allowed the system to output keyphrases as opposed to just the singular keywords of the unigram study.

For the n -gram study, the words from the source document were split into a number of n -gram lists, from unigrams up to 7-grams. For all of the lists the entries overlapped so that all combinations of words from the text were included. E.g., if the source text were “The quick fox jumped” then the bigrams would be “The quick”, “quick fox”, and “fox jumped” and the trigrams would be “The quick fox”, and “quick fox jumped”. For each document, the results of each of the n -grams were combined and considered together to determine the overall output.

- For each n -gram in the thesaurus, compare the n -gram to the associated synonyms.
- For each synonym that matches, add the word to a list, and increase its frequency value by the value of the n -gram.
- Sort the list by frequency and output the top r ranked items (in this study, r was chosen to be 5).

This algorithm is shown in Figure 4.

D. The clustering study

Examining the results of the n -gram study (as discussed in Section V below) revealed that only the highest frequency “group” or cluster of synonyms was being matched, and as such the clustering algorithm attempts to extend the n -gram algorithm to group the keyphrases into “clusters”. It achieves this by finding the keyphrases that are of a similar theme and returning a single keyphrase for that group.

For example, the word “recovery” can mean either “acquisition” or “taking” [2]. The base system therefore could return multiple versions of the same concept as keyphrases. By clustering the results, the attempt was to prevent a single, “popular”, concept dominating and allow the other themes to be represented. The method for this was:

- For each n -gram in the thesaurus, compare the n -gram to the associated synonyms
- For each synonym that matches, add the word to a list, and increase its frequency value by the value of the n -gram divided by the number of associated synonyms
- Then, for each Key entry in the thesaurus check to see if the frequency is equal to the highest frequency value in the found in the preceding step.
- For each synonym entry associated with the Key, add the synonym to a second list of words and increase its value by one.
- Sort the second list by frequency and output the top r ranked items (in this study, r was chosen to be 5).

This algorithm is shown in Figure 5.

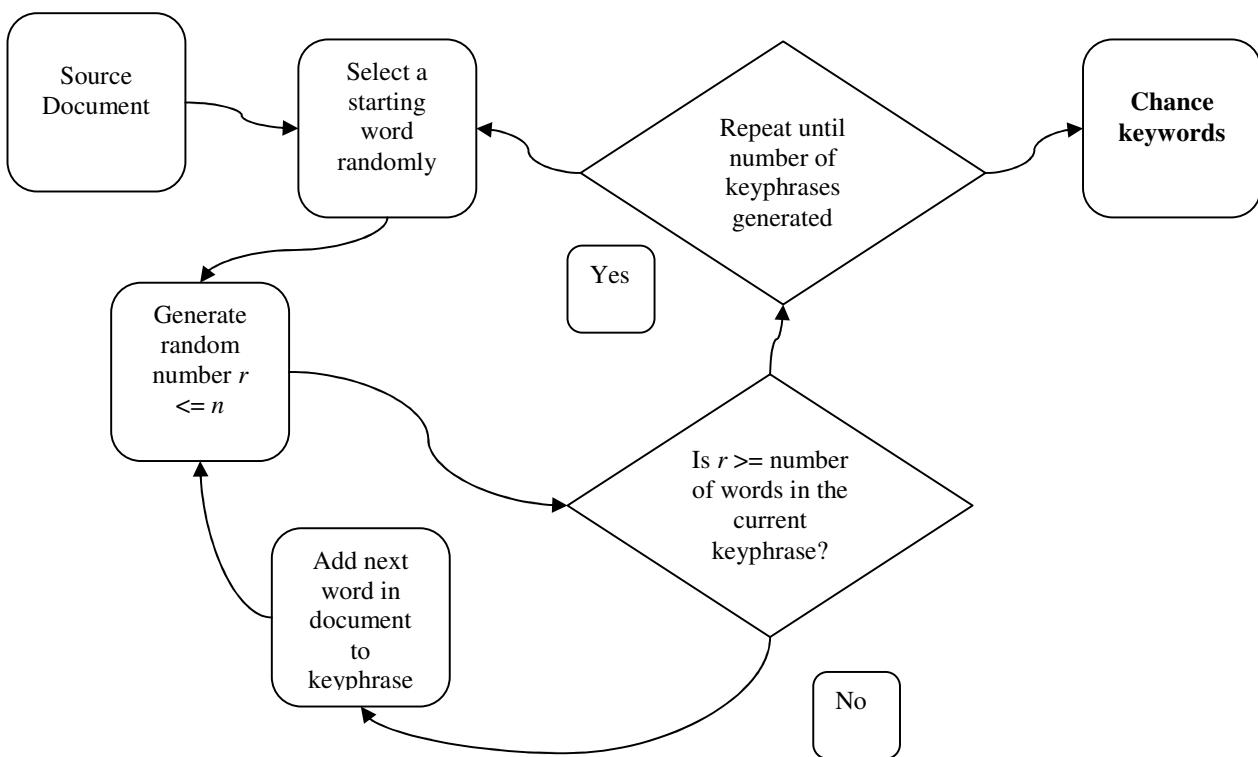


Figure 3. Chance algorithm

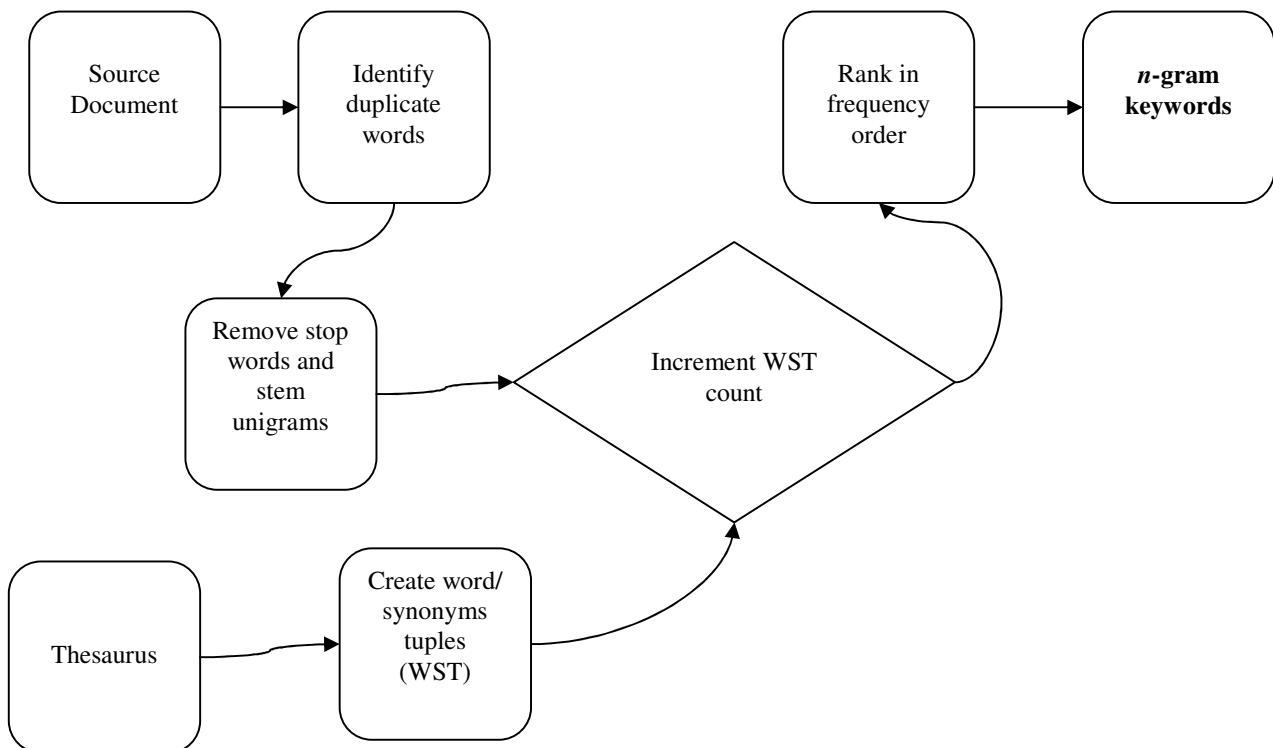


Figure 4. n -gram algorithm

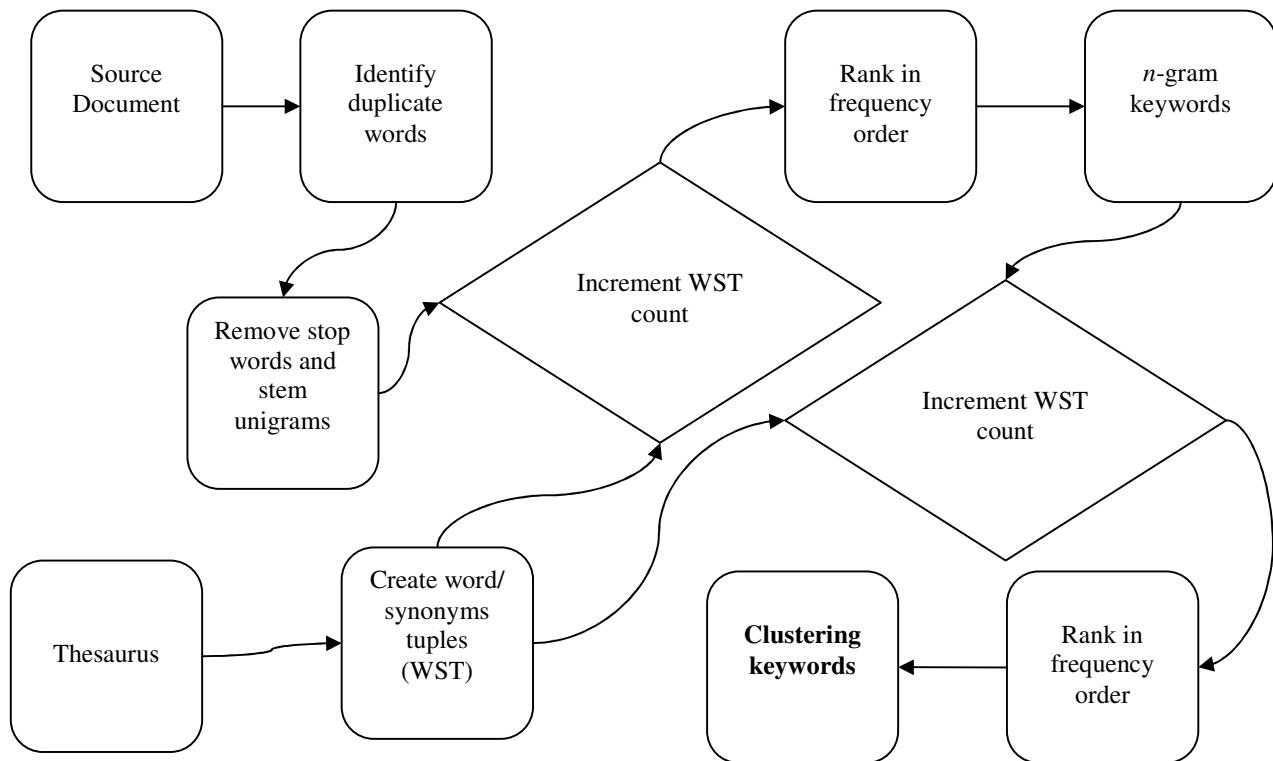


Figure 5. Clustering algorithm

IV. RESULTS

The results of these four studies are shown below. For each of the e-journals used, the authors of each paper in the journal had supplied an accompanying list of keyphrases summarising the content of that paper. These were therefore leveraged to provide a method of automatically evaluating the results of the work presented here.

For every paper, a match was recorded if at least one author-supplied keyphrase was a substring of, a superstring of, or exactly equal to a system-supplied keyword. This naïve text-matching approach would match the word “know” with both the words “know” and “knowledge”.

For all of the tables the following explanations of each column apply. The ‘Journal’ column lists the five e-journals from ACI [20], and the ‘Papers’ column lists the number of papers in that corpus. The number ‘Matched’ is the number of papers in that journal that recorded a match, and ‘Percentage’ is the percentage number of papers in that journal that were considered a match. Where it appears, ‘Increase’ is the numerical value by which the percentage match has increased over the results of the chance study – i.e. if the match percentage was 5% in the chance study and 11% in n-gram study that would be an increase of 6.

A. Chance Study

The chance results showed almost no keyphrases being produced that matched the authors. The results can be seen in Table I.

TABLE I. CHANCE RESULTS

Journal	Papers	Matched	Percentage
EJBRM	72	0	0.00%
EJEG	101	2	1.98%
EJEL	112	0	0.00%
EJISE	91	1	1.11%
EJKM	110	5	4.81%
Average			1.58%

B. Baseline System

Table II, Table III, and Table IV show the baseline results for the study. The increase measures the performance compared to the results from Table I. The average percentage correct was 5.80%, an increase of 4.22 over the chance results from Table I.

TABLE II. BASE LINE ROGET RESULTS

Journal	Papers	Matched	Percentage	Increase
EJBRM	72	4	5.56%	5.56
EJEG	101	3	2.97%	0.99
EJEL	112	18	16.07%	16.07
EJISE	91	7	7.69%	6.58
EJKM	110	19	17.27%	12.46
Average		9.91%	8.33	

TABLE III. BASE LINE WORDNET RESULTS

Journal	Papers	Matched	Percentage	Increase
EJBRM	72	0	0.00%	0.00
EJEG	101	3	2.97%	0.99
EJEL	112	0	0.00%	0.00
EJISE	91	1	1.11%	0.00
EJKM	110	6	5.77%	0.64
Average		1.90%	0.32	

TABLE IV. BASE LINE MOBY RESULTS

Journal	Papers	Matched	Percentage	Increase
EJBRM	72	5	6.94%	6.94
EJEG	101	4	3.96%	1.98
EJEL	112	3	2.68%	2.68
EJISE	91	9	9.89%	8.78
EJKM	110	5	4.55%	-0.26
Average		5.60%	4.02	

C. The n-gram study

The n-gram results showed a small improvement over the baseline, as can be seen in Table V, Table VI, and Table VII. The increase measures the performance compared to the results from Table I. The average percentage correct was 23.59%, an increase of 22.01 over the chance results from Table I.

TABLE V. RESULTS OF ROGET N-GRAM STUDY

Journal	Papers	Matched	Percentage	Increase
EJBRM	72	16	24.62%	24.62
EJEG	101	21	20.79%	18.81
EJEL	112	54	49.54%	19.54
EJISE	91	27	30.00%	28.89
EJKM	110	70	67.31%	62.50
Average		38.45%	30.87	

TABLE VI. RESULTS OF WORDNET N-GRAM STUDY

Journal	Papers	Matched	Percentage	Increase
EJBRM	72	9	13.85%	13.85
EJEG	101	17	16.83%	14.85
EJEL	112	12	11.01%	11.01
EJISE	91	8	8.89%	7.78
EJKM	110	15	14.42%	9.61
Average		13.00%	11.42	

TABLE VII. RESULTS OF MOBY N-GRAM STUDY

Journal	Papers	Matched	Percentage	Increase
EJBRM	72	17	23.61%	23.61
EJEG	101	18	17.82%	15.84
EJEL	112	18	16.07%	16.07
EJISE	91	19	20.88%	19.77
EJKM	110	20	18.18%	13.37
Average		19.31%	17.73	

D. The clustering study

The clustering results show a reasonable improvement over the n-gram results and a significant increase over the chance results, as can be seen in Table VIII, Table IX, and Table X. The increase measures the performance compared to the results from Table I. The average percentage correct was 45.75%, an increase of 44.17 over the chance results from Table I.

TABLE VIII. RESULTS OF ROGET CLUSTERING STUDY

Journal	Papers	Matched	Percentage	Increase
EJBRM	72	31	43.06%	43.06
EJEG	101	73	72.28%	70.30
EJEL	112	77	68.75%	68.75
EJISE	91	46	50.55%	49.44
EJKM	110	94	85.45%	80.64
Average		64.02%	62.44	

TABLE IX. RESULTS OF WORDNET CLUSTERING STUDY

Journal	Papers	Matched	Percentage	Increase
EJBRM	72	41	63.08%	63.08
EJEG	101	69	68.32%	66.34
EJEL	112	37	33.94%	33.94
EJISE	91	38	42.22%	41.11
EJKM	110	57	54.81%	50.00
Average		52.47%	50.89	

TABLE X. RESULTS OF MOBY CLUSTERING STUDY

Journal	Papers	Matched	Percentage	Increase
EJBRM	72	16	22.22%	22.22
EJEG	101	21	20.79%	18.81
EJEL	112	20	17.86%	17.86
EJISE	91	20	21.98%	20.87
EJKM	110	23	20.91%	16.10
Average		20.75%	19.17	

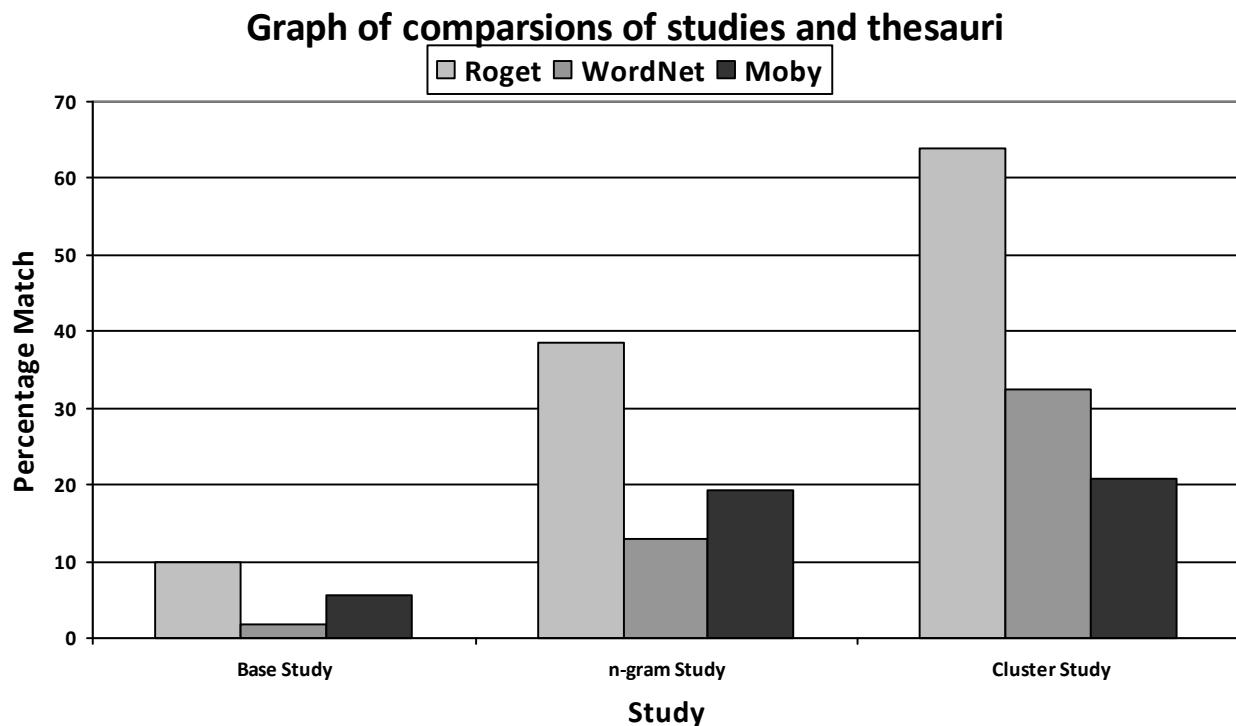


Figure 6. Graph of Studies/Percentage

V. DISCUSSION

The results show that using n -grams on their own produces a significant improvement over both chance and the baseline study (an average over the three thesauri of 23.59%). This shows that this method of using a thesaurus to group words into their conceptual clusters has potential to produce useful outputs.

However, the results did not vary when the number of n -grams was changed (ranging between 1 and 7) but the number of outputs r was maintained (this section was only tested on for the *WordNet* thesaurus). A possible explanation for this would be only the highest frequency group of synonyms is being matched by the author keywords.

Therefore, the algorithm was extended to include the clustering algorithm, which in turn produced a further, and significant, improvement (an average of 45.75% across the three thesauri). The results are shown in Figure 6 grouped by study, and clearly show that each addition to the study improved on the average result, and that in all studies the *Roget* thesaurus outperformed the rest. This is confirmed by Figure 7, which shows the same results grouped instead by thesauri.

In addition to the issues found in the n -gram study further improvement on the results seems to be unlikely due to issues with the mechanism for confirming a match – author

keywords. Some of the keywords submitted by the authors of the papers in the corpus may be tags instead of keywords. These can display meta-data that can often be irrelevant to the understanding of the document. An example seen in the corpus was the keyword “University of Birmingham” because the author of that paper worked there. This is valid as a tag but as a keyword, as it does not indicate a topic or a theme to which the document holds (other than in a rare case where the paper is about the University of Birmingham). This therefore lowers the chances of keyphrases being matched as the comparison data is filled with ‘noise’.

The synonyms are currently analysed context-free, and thus for a word with multiple meanings (e.g., “recovery” can mean “acquisition”, “improvement”, or “restoration” [2]) every occurrence of that word is treated the same. This means that a document equally about “improvement” and “restoration” could end up with the theme of “recovery” which (while a correct assumption) may not give the right meaning.

A. Thesauri outcomes

The results from the various studies all show that on average the *Roget’s Thesaurus* outperforms *WordNet*, which in turn outperforms *Moby’s Thesaurus*.

Appendix A contains a sample entry from each thesaurus for the word “question” (as an example). As can be seen, the *Roget* entry is the shortest and the *Moby* entry the longest and most comprehensive. As a thesaurus, *Roget* has 55,000 entries, *Moby* has 30,000, and *WordNet* has 5,000.

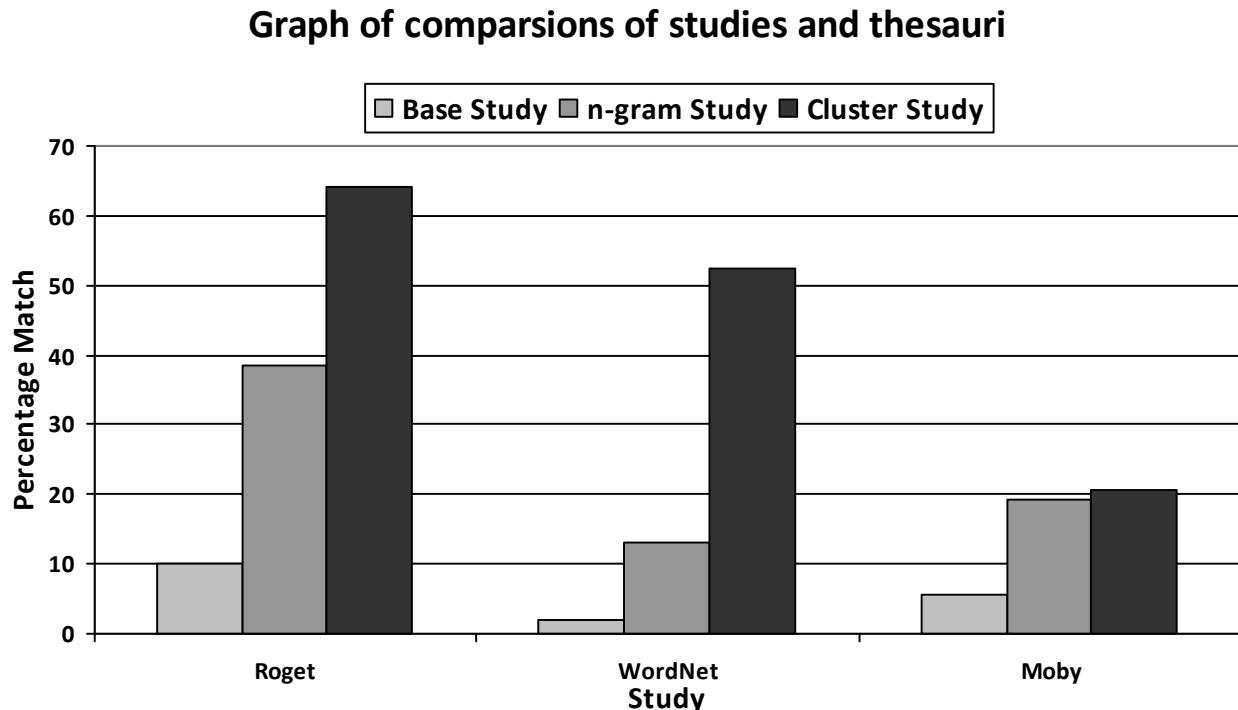


Figure 7. Graph of Thesauri/Percentage

The *Moby* and *WordNet* thesaurus entries are both newer (less than fifteen years old) than their counterpart *Roget* is, and consequently contain modern phrases such as “sixty-four dollar question” (see Appendix A). Yet, in spite of this, they perform worse than the one hundred year old thesaurus.

McHale [22] compares *WordNet* and *Roget* for measuring semantic similarity, and concludes that due to the combined relative uniformity of the hierarchy in *Roget* and the broader allowed set of semantic relationships, that it seems better at capturing “the popular similarity of isolated word pairs”. This potentially allows it to find more words around a single concept, compared to the other thesauri studied, which work in smaller concept-circles.

VI. CONCLUSION AND FURTHER WORK

The approach to synonym analysis developed in this paper shows good results for the test corpora used and potential for future study. Further study is required to compare the system to ones developed in similar areas, but this should provide a solid framework for taking the project forward.

The results, as mentioned in Section 0, show that the number of *n*-grams used does not affect the outcome of the system – all that matters is using the synonyms. This does not, however, mean that the keywords produced may not be more useful to the user, as they could be different enough not to match the success criteria but still relevant.

The results themselves were evaluated against the keywords submitted by the authors of the papers. *TagAssist*

[4] showed that in 54.15% of cases, author keywords were judged as being inappropriate for the work with which they were associated. Therefore, when interpreting the results (which averaged around 60% matches) it should be remembered that they are produced by matching the output against the author keywords, which may be less than perfect for the task. A new method of evaluating the results is therefore required.

Another area of further work is to conduct more experiments to determine what differences there are between the thesauri, and what impacts the differences have on the results. When compared, results from *Roget*'s thesaurus produced better results than *WordNet* and *Moby*, but it is not clear at this stage why that is the case. It is possible, for example, that each of the thesauri is suited to a certain subject corpora (e.g., a medical corpus vs. a computer science corpus). Therefore, more experiments will need to be run with different corpora to ascertain if this is the case, or if the *Roget*'s thesaurus is simply better suited to this application than the other two.

In addition, given the difference in size of each thesaurus a further area of study would be to attempt to make a single thesaurus that only contains the words found in all three and to see how well that thesaurus compares to the existing results. In a similar vein to this, another study would be to combine all three thesauri into a single but larger thesaurus and compare that to the existing results as well as to the version with reduced entries.

APPENDIX A

This appendix includes the entries from the three thesauri for the word “question”.

A. Roget entry for “question”

Question

- inquiry, irreligion, unbelief doubt

Taken from [2]

B. WordNet entry for “question”

Question

- inquiry, query, interrogation, interrogate

Taken from [15]

C. Moby entry for “question”

Question

- Chinese puzzle, Parthian shot, Pyrrhonism, absurd, address, affirmation, agonize over, allegation, answer, apostrophe, apprehension, approach, ask, ask a question, ask about, ask questions, assertion, assuredly, at issue, averment, awake a doubt, baffling problem, basis, be at sea, be curious, be diffident, be doubtful, be dubious, be sceptical, be uncertain, beat about, bill, blind bargain, bone of contention, borderline case, brain twister, bring into question, burden, burn with curiosity, calendar, call in question, case, catechism, catechize, certainly, challenge, chance, chapter, clause, comment, communicate with, companion bills amendment, concern, confusion, contact, contest, contingency, correspond, crack, cross-interrogatory, cross-question, crossword puzzle, crux, debatable, debating point, declaration, definitely, demand, demurral, demurrer, dictum, difficulty, diffidence, dig around for, dig up, dispute, distrust, distrustfulness, double contingency, doubt, doubtful, doubtfulness, doubtlessly, dragnet clause, dubiety, dubiousness, enacting clause, enigma, enigmatic question, enquiry, escalator clause, essence, establish connection, examine, exclamation, expression, feel unsure, feeler, focus of attention, focus of interest, gamble, gape, gawk, get to, gist, greet with scepticism, greeting, grill, grope, guess, half believe, half-belief, harbour suspicions, have reservations, head, heading, hold-up bill, impossible, in doubt, in question, inconceivable, indubitably, inquire, inquire of, inquiry, insupportable, interjection, interpolate, interrogate, interrogation, interrogative, interrogatory, interview, issue, jigsaw puzzle, joker, knot, knotty point, leader, leading question, leeriness, living issue, main point, maintain connection, make advances, make contact with, make inquiry, make overtures,

make up to, matter, matter in hand, meat, mention, mind-boggler, misdoubt, misgive, misgiving, mistrust, mistrustfulness, moot point, motif, motion, motive, mystery, nose around for, nose out, note, nut, nut to crack, objection, observation, omnibus bill, open question, peer, perplexed question, perplexity, phrase, piece of guesswork, point, point at issue, point in question, poser, position, preposterous, privileged question, problem, pronouncement, propose a question, proposition, propound a question, protest, proviso, pump, put queries, puzzle, puzzle over, puzzlement, puzzler, query, question, question at issue, question mark, questionable, questioning, quiz, quodlibet, raise, raise a question, reach, reflection, relate to, remark, remonstrance, remonstration, reply to, require an answer, respond to, rider, ridiculous, rubber, rubberneck, rubric, saving clause, say, saying, scruple, scrupulousness, seek, self-doubt, sentence, shadow of doubt, sight-unseen transaction, sixty-four dollar question, scepticalness, scepticism, smell a rat, sound out, stare, statement, sticker, stumper, subject, subject matter, subject of thought, subjoinder, substance, suspect, suspicion, suspiciousness, test, text, theme, thought, thrash about, throw doubt upon, topic, toss-up, total scepticism, touch and go, tough proposition, treat with reserve, trial balloon, uncertainty, undecided issue, under consideration, undoubtedly, unthinkable, utterance, vexed question, wager, want to know, wariness, why, wonder, wonder about, wonder whether, word, worm out of

Taken from [19]

ACKNOWLEDGMENT

The authors would like to thank the School of Systems Engineering for the studentship, which enabled this project, and the contributions from the reviewers to this paper.

REFERENCES

- [1] R. Hussey, S. Williams, and R. Mitchell. 2011. “Keyphrase Extraction by Synonym Analysis of n -grams for E-Journal Classification”, eKNOW, Proceedings of The Third International Conference on Information, Process, and Knowledge Management, pp. 83-86. Gosier, Guadeloupe/France.
http://www.thinkmind.org/index.php?view=article&articleid=eknow_2011_4_30_60053 [Last accessed: 23 January 2012]
- [2] P.M. Roget. 1911. “Roget’s Thesaurus of English Words and Phrases (Index)”. <http://www.gutenberg.org/etext/10681> [Last accessed: 23 January 2012]
- [3] E. Frank, G.W. Paynter, I.H. Witten, C. Gutwin, and C.G. Nevill-Manning. 1999. “Domain-Specific Keyphrase Extraction”, Proceedings 16th International Joint Conference on Artificial Intelligence, pp. 668-673. San Francisco, CA Morgan Kaufmann Publishers.

- [4] S.C. Sood, S.H. Owsley, K.J. Hammond, and L. Birnbaum. 2007. "TagAssist: Automatic Tag Suggestion for Blog Posts". Northwestern University. Evanston, IL, USA. <http://www.icwsm.org/papers/2--Sood-Owsley-Hammond-Birnbaum.pdf> [Last accessed: 23 January 2012]
- [5] Technorati. 2006. "Technorati". <http://www.technorati.com> [Last accessed: 23 January 2012]
- [6] J. Goldstein, M. Kantrowitz, V. Mittal, and J. Carbonell. 1999. "Summarising Text Documents: Sentence Selection and Evaluation Metrics", ACM, pp. 121–128. Language Technologies Institute, Carnegie Mellon University, Pittsburgh, USA.
- [7] C. Li, J. Wen, and H. Li. 2003. "Text Classification Using Stochastic Keyword Generation", Twentieth International Conference on Machine Learning (ICML), pp 464–471. Washington DC. <https://www.aaai.org/Papers/ICML/2003/ICML03-062.pdf> [Last accessed: 23 January 2012]
- [8] K. Barker and N. Cornacchia. 2000. "Using Noun Phrase Heads to Extract Document Keyphrases", AI '00: Proceedings of the 13th Biennial Conference of the Canadian Society on Computational Studies of Intelligence. pp. 40–52). London.
- [9] J. Wermter and U. Hahn. 2005. "Paradigmatic Modifiability Statistics for the Extraction of Complex Multi- Word Terms". Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing (HLT/EMNLP) pp. 843–850. Vancouver Association for Computational Linguistics.
- [10] K. Frantzi, S. Ananiadou, and H. Mimaz. 2000. "Automatic Recognition of Multi-Word Terms: the C-value/NC-value Method", International Journal on Digital Libraries , 3 (2), pp. 117-132.
- [11] D. Maynard and S. Ananiadou. 2000. "TRUCKS: a model for automatic multi-word term recognition". Journal of Natural Language Processing, 8 (1), pp. 101-125.
- [12] D. Maynard and S. Ananiadou. 1999. "Term extraction using a similarity-based approach". Recent Advances in Computational Terminology, pp. 261–278.
- [13] A. Joshi and R. Motwani. 2006. "Keyword Generation for Search Engine Advertising", IEEE International Conference on Data Mining, pp. 490–496.
- [14] S. Scott and S. Matwin. 1998. "Text Classification Using WordNet Hypernyms", Proceedings of the Association for Computational Linguistics, pp. 38–44.
- [15] G.A. Miller, C. Fellbaum, R. Tengi, P. Wakefield, and H. Langone. 2005. "WordNet". Princeton University. <http://WordNet.princeton.edu> [Last accessed: 23 January 2012]
- [16] D. Greenhaus. 2002. "DigiTrad - Digital Tradition Folk Song Server". <http://www.mudcat.org/download.cfm> [Last accessed: 23 January 2012]
- [17] Reuters. 1987. "Reuters-21578 Text Categorisation Collection". <http://kdd.ics.uci.edu/databases/reuters21578/reuters21578.htm> [Last accessed: 23 January 2012]
- [18] B. Wei, C. Zhang, and M. Ogihara. 2007. "Keyword Generation for Lyrics", Austrian Computer Society (OCG). Comp. Sci. Dept., U. Rochester, USA. http://ismir2007.ismir.net/proceedings/ISMIR2007_p121_wei.pdf [Last accessed: 23 January 2012]
- [19] G. Ward. 2000. "Moby Project - Thesaurus". <http://icon.shef.ac.uk/Moby/mthes.html> [Last accessed: 11 July 2011]
- [20] Academics Conferences International. 2009. "ACI E-Journals". <http://academic-conferences.org/ejournals.htm> [Last accessed: 23 January 2012]
- [21] M.F. Porter. 1980. "An algorithm for suffix stripping", Program, 14(3) pp. 130–137.
- [22] M.L. McHale. 1998. "A Comparison of WordNet and Roget's Taxonomy for Measuring Semantic Similarity", <http://acl.ldc.upenn.edu/W/W98/W98-0716.pdf> [Last accessed: 23 January 2012]



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
- ◆ issn: 1942-2679

International Journal On Advances in Internet Technology

- ◆ ICDS, ICIW, CTRQ, UBICOMM, ICSNC, AFIN, INTERNET, AP2PS, EMERGING
- ◆ issn: 1942-2652

International Journal On Advances in Life Sciences

- ◆ eTELEMED, eKNOW, eL&mL, BIODIV, BIOENVIRONMENT, BIOGREEN, BIOSYSCOM, BIOINFO, BIOTECHNO
- ◆ issn: 1942-2660

International Journal On Advances in Networks and Services

- ◆ ICN, ICNS, ICIW, ICWMC, SENSORCOMM, MESH, CENTRIC, MMEDIA, SERVICE COMPUTATION
- ◆ 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
- ◆ issn: 1942-2628

International Journal On Advances in Systems and Measurements

- ◆ ICQNM, ICONS, ICIMP, SENSORCOMM, CENICS, VALID, SIMUL
- ◆ issn: 1942-261x

International Journal On Advances in Telecommunications

- ◆ AICT, ICDT, ICWMC, ICSNC, CTRQ, SPACOMM, MMEDIA
- ◆ issn: 1942-2601