Tiziana Margaria
Bernhard Steffen (Eds.)

LNCS 6415

Leveraging Applications of Formal Methods, Verification, and Validation

4th International Symposium
on Leveraging Applications, ISoLA 2010
Heraklion, Crete, Greece, October 2010, Proceedings, Part I

Springer

Lecture Notes in Computer Science 6415

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Tiziana Margaria Bernhard Steffen (Eds.)

Leveraging Applications of Formal Methods, Verification, and Validation

4th International Symposium
on Leveraging Applications, ISoLA 2010
Heraklion, Crete, Greece, October 18-21, 2010
Proceedings, Part I

Springer

Volume Editors

Tiziana Margaria
University of Potsdam
August-Bebel-Str. 89
14482 Potsdam
Germany
E-mail: margaria@cs.uni-potsdam.de

Bernhard Steffen
TU Dortmund University
Otto-Hahn-Str. 14
44227 Dortmund
Germany
E-mail: steffen@cs.tu-dortmund.de

Library of Congress Control Number: 2010936699

CR Subject Classification (1998): F.3, D.2.4, D.3, C.2-3, D.2, I.2

LNCS Sublibrary: SL 1 – Theoretical Computer Science and General Issues

ISSN 0302-9743
ISBN-10 3-642-16557-5 Springer Berlin Heidelberg New York
ISBN-13 978-3-642-16557-3 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper 06/3180

Preface

This volume contains the conference proceedings of the 4th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation, ISoLA 2010, which was held in Greece (Heraklion, Crete) October 18–21, 2010, and sponsored by EASST.

Following the tradition of its forerunners in 2004, 2006, and 2008 in Cyprus and Chalchidiki, and the ISoLA Workshops in Greenbelt (USA) in 2005, in Poitiers (France) in 2007, and in Potsdam (Germany) in 2009, ISoLA 2010 provided a forum for developers, users, and researchers to discuss issues related to the adoption and use of rigorous tools and methods for the specification, analysis, verification, certification, construction, testing, and maintenance of systems from the point of view of their different application domains. Thus, the ISoLA series of events serves the purpose of bridging the gap between designers and developers of rigorous tools, and users in engineering and in other disciplines, and to foster and exploit synergetic relationships among scientists, engineers, software developers, decision makers, and other critical thinkers in companies and organizations. In particular, by providing a venue for the discussion of common problems, requirements, algorithms, methodologies, and practices, ISoLA aims at supporting researchers in their quest to improve the utility, reliability, flexibility, and efficiency of tools for building systems, and users in their search for adequate solutions to their problems.

The program of the symposium consisted of special tracks devoted to the following hot and emerging topics:

- Emerging services and technologies for a converging telecommunications/Web world in smart environments of the Internet of Things
- Learning techniques for software verification and validation
- Modeling and formalizing industrial software for verification, validation and certification
- Formal methods in model-driven development for service-oriented and cloud computing
- Tools in scientific workflow composition
- New challenges in the development of critical embedded systems—an "aeromotive" perspective
- Web science
- Leveraging formal methods through collaboration
- Resource and timing analysis
- Quantitative verification in practice
- Worst case traversal time (WCTT)
- Model transformation and analysis for industrial scale validation
- Certification of software-driven medical devices
- Formal languages and methods for designing and verifying complex engineering systems

- CONNECT: status and plan
- EternalS: mission and roadmap

and five co-located events

- Graduate/postgraduate course on "Soft Skills for IT Professionals in Science and Engineering"
- RERS—challenge on practical automata learning
- IT Simply Works—editorial meeting (ITSy)
- CONNECT internal meeting
- EternalS Task Force meetings

We thank the Track organizers, the members of the Program Committee and their subreferees for their effort in selecting the papers to be presented.

Special thanks are due to the following organization for their endorsement: EASST (European Association of Software Science and Technology), and our own institutions—the TU Dortmund, and the University of Potsdam.

August 2010

Tiziana Margaria
Bernhard Steffen

Organization

Committees

Symposium Chair

Tiziana Margaria	University of Potsdam, Germany

Program Chair

Bernhard Steffen	TU Dortmund, Germany

Program Committee

Yamine Ait Ameur	LISI/ENSMA, France
Frédéric Boniol	IRIT/ENSEEIHT, France
Anne Bouillard	ENS Cachan, France
Marc Boyer	ONERA, France
Karin Breitman	PUC-Rio, Brazil
Marco Antonio Casanova	PUC-Rio, Brazil
Samarjit Chakraborty	TU München, Germany
Noel Crespi	Institut Telecom, France
Rémi Delmas	ONERA, France
Howard Foster	City University London, UK
Pierre-Loïc Garoche	ONERA, France
Dimitra Giannakopoulou	CMU/NASA Ames, USA
Stefania Gnesi	ISTI-CNR, Pisa, Italy
Kevin Hammond	University of St Andrews, UK
Boudewijn Haverkort	ESI, The Netherlands
Michael Hinchey	LERO, Ireland
Valérie Issarny	INRIA, France
Visar Januzaj	TU Darmstadt, Germany
He Jifeng	East China Normal University, China
Joost-Pieter Katoen	RWTH Aachen University, Germany
Joost Kok	Leiden University, The Netherlands
Jens Knoop	Vienna University of Technology, Austria
Stefan Kugele	TU München, Germany
Anna-Lena Lamprecht	TU Dortmund, Germany
Kim G. Larsen	Aalborg University, Denmark
Boris Langer	Diehl Aerospace, Germany

Mark Lawford	McMaster University, Canada
Gyu Myoung Lee	Institut Télécom, France
Björn Lisper	Mälardalen University, Sweden
Zhiming Liu	UNU-IIST, Macao
Tom Maibaum	McMaster University, Canada
Steven Martin	LRI, France
Dominique Mery	University Nancy, France
Pascal Montag	Daimler AG, Germany
Alessandro Moschitti	University of Trento, Italy
Corina Pasareanu	CMU/NASA Ames, USA
Alexander K. Petrenko	ISPRAS, Moscow, Russia
Abhik Roychoudhury	NUS, Singapore
Christian Schallhart	Oxford University, UK
Jean-Luc Scharbarg	IRIT, France
Amal Seghrouchni	University Pierre and Marie Curie, France
Laura Semini	Pisa University, Italy
Giovanni Stea	Pisa University, Italy
Eric Thierry	ENS Lyon, France
Helmut Veith	Vienna University of Technology, Austria
Alan Wassyng	McMaster University, Canada
Virginie Wiels	ONERA, France
Mark D. Wilkinson	Heart and Lung Institute, and Canada
Rostislav Yavorskiy	Microsoft UK/Moscow, Russia
Lenore Zuck	University of Illinois at Chicago, USA

Table of Contents – Part I

New Challenges in the Development of Critical Embedded Systems – An "aeromotive" Perspective

Formal Languages and Methods for Designing and Verifying Complex Embedded Systems

Worst-Case Traversal Time (WCTT)

Tools in Scientific Workflow Composition

Emerging Services and Technologies for a Converging Telecommunications / Web World in Smart Environments of the Internet of Things

Web Science

Model Transformation and Analysis for Industrial Scale Validation

Learning Techniques for Software Verification and Validation

Table of Contents – Part II

EternalS: Mission and Roadmap

Formal Methods in Model-Driven Development for Service-Oriented and Cloud Computing

Quantitative Verification in Practice

CONNECT: Status and Plans

Certification of Software-Driven Medical Devices

Modeling and Formalizing Industrial Software for Verification, Validation and Certification

Resource and Timing Analysis

New Challenges in the Development of Critical Embedded Systems—An "aeromotive" Perspective

Visar Januzaj[2], Stefan Kugele[3], Boris Langer[5], Christian Schallhart[4], and Helmut Veith[1]

[1] Technische Universität Wien
AB Formal Methods in Systems Engineering
Favoritenstraße 9, 1040 Wien, Austria
veith@forsyte.at
[2] Technische Universität Darmstadt, Fachbereich Informatik,
FG Formal Methods in Systems Engineering,
Hochschulstr. 10, 64289 Darmstadt, Germany
januzaj@forsyte.de
[3] Technische Universität München, Institut für Informatik,
Boltzmannstr. 3, 85748 Garching bei München, Germany
kugele@in.tum.de
[4] Oxford University Computing Laboratory
Wolfson Building, Parks Road, OX1 3QD Oxford, United Kingdom
christian.schallhart@comlab.ox.ac.uk
[5] Diehl Aerospace GmbH, Business Line Avionics
An der Sandelmühle 13, 60439 Frankfurt, Germany
boris.langer@diehl-aerospace.de

During the last decades, embedded systems have become increasingly important in highly safety-critical areas such as power plants, medical equipment, cars, and aeroplanes. The automotive and avionics domains are prominent examples of classical engineering disciplines where conflicts between costs, short product cycles and legal requirements concerning dependability, robustness, security, carbon footprint and spatial demands have become a pressing problem. This setting necessitates a disciplined, rigorous approach to systems engineering which is able to find good trade-offs between complex and conflicting requirements. A successful large-scale project thus needs to be based on a well-defined process model. The functional safety standard ISO 26262 ("Road vehicles – Functional safety"), which is ubiquitous in the development of automotive software, defines such a process along with the involved activities. Dr. Bernhard Schätz from fortiss GmbH, Germany, will focus in his invited talk *Certification of embedded software—Impact of ISO DIS 26262 in the Automotive Domain* on the chances and questions arising from ISO 26262 in practice.

In such large-scale safety-critical systems, the strong requirements for system quality and absence of errors need to be addressed in a way that accounts for the high complexity of distributed systems. Model-driven development (MDD) is considered as the most promising technique to tackle this complexity and is

T. Margaria and B. Steffen (Eds.): ISoLA 2010, Part I, LNCS 6415, pp. 1–2, 2010.

accepted as a de-facto standard in the industry. In the article *Seamless Model-driven Development put into Practice* Haberl et al. propose three important cornerstones which a proper MDD approach has to support, namely, a well-defined modelling theory, a defined process, and an integrated tool(-chain). To support seamless modelling, all tools are built upon the same product data model, facilitating the modelling of artefacts on different levels of abstraction.

Besides a sophisticated modelling environment, powerful analysis techniques are essential for system development. The analysis of general timing aspects and of temporal system behaviour are crucial to guarantee the correct, safe and intended operation of distributed real-time systems. Plantec et al. contribute to this field in their paper *Enforcing Applicability of Real-time Scheduling Theory Feasibility Tests with the use of Design-Patterns*. They propose a technique to help designers in checking whether an architectural system model specified in AADL is compliant with the real-time scheduling theory. Early performance verification of architectural models is possible by using a set of architecture design-patterns.

The challenge of early timing analysis is also addressed by Holzer et al. in their paper *Timely Time Estimates*. The authors propose a framework capable to supply engineers with execution time estimates at an early development stage. Their approach allows developers to ask for time estimates in a demand-driven manner on the level of C code. Using a repository of benchmarks and results, they avoid the need for a detailed and very expensive hardware model at an early stage.

Due to the increased functional range of automotive software as well as the increased complexity of the technology itself—for instance, in multi-core systems—a posteriori analysis of software models has become very challenging. Kirner et al. propose in *Compiler-Support for Robust Multi-Core Computing* to tackle this problem by developing an enhanced compiler. They propose to implement support for software-controlled robustness for multi-core architectures within the SaC research compiler. Its functional input language simplifies classic questions such as fault-recovery.

In summary, this track is intended to foster interaction between the different technologies involved in the development of "aeromotive" systems. The contributions collected here range from modelling and analysis to technical aspects and certification.

Certification of Embedded Software – Impact of ISO DIS 26262 in the Automotive Domain

Bernhard Schätz

fortiss gGmbH
Guerickestr. 25, 80805 München, Germany
schaetz@fortiss.org

Abstract. The publication of the ISO 26262 ("Road vehicles – Functional safety") as Draft International Standard (DIS) and its expected release as international standard in 2011 has a substantial impact on the development of automotive software. By defining the current state of technique for the development of safe automotive software, the lack of or inadequate use of these techniques has severe legal consequences.

Like its ancestor, IEC 61508, as a process standard the ISO DIS 26262 defines artifacts and activities of the development process; consequently, Part 6 of the ISO standard ("Product development: Software Level") defines the artifacts and activities for requirements specification, architectural design, unit implementation and testing, as well as system integration and verification. Depending on the hazard analysis and risk assessment, and on the resulting Automotive Safety Integrity Level (ASIL) of the function under development, the standard, e.g., prescribes the use of (semi)formal methods for the verification of requirements, (semi-)formal notations for software design, the use of control and data flow analysis techniques, static and semantic code analysis, the use of test case generation, or in-the-loop verification mechanisms. Furthermore, the standard specifically acknowledges the application of model-based development in automotive software engineering.

Currently, several of these rather advanced techniques are only required for higher safety integrity levels. Consequently, even though embedded software has become the leading innovation factor in automotive applications, many highly safety-critical automotive functionalities are only reluctantly implemented with software-based solutions. Here, by advancing the applicability and scalability of these advanced technologies and providing support in form of qualified tool chains, a substantial change in the development of automotive software can be achieved, allowing not only to virtualize and thus substitute physical solutions of automotive functions (e.g., X-by-wire solutions), but also to implement a new range of functionalities (e.g., autonomic driving).

T. Margaria and B. Steffen (Eds.): ISoLA 2010, Part I, LNCS 6415, p. 3, 2010.

Enforcing Applicability of Real-Time Scheduling Theory Feasibility Tests with the Use of Design-Patterns

Alain Plantec[1], Frank Singhoff[1], Pierre Dissaux[2], and Jérôme Legrand[2]

[1] LISyC, University of Brest, UEB, 20 av. Le Gorgeu, 29238 Brest, France
{alain.plantec,frank.singhoff}@univ-brest.fr
[2] Ellidiss Technologies, 24 quai de la douane, 29200 Brest, France
{pierre.dissaux,jerome.legrand}@ellidiss.com

Abstract. This article deals with performance verifications of architecture models of real-time embedded systems. We focus on models verified with the real-time scheduling theory. To perform verifications with the real-time scheduling theory, the architecture designers must check that their models are compliant with the assumptions of this theory. Unfortunately, this task is difficult since it requires that designers have a deep understanding of the real-time scheduling theory. In this article, we investigate how to help designers to check that an architecture model is compliant with this theory. We focus on feasibility tests. Feasibility tests are analytical methods proposed by the real-time scheduling theory. We show how to explicitly model the relationships between an architectural model and feasibility tests. From these models, we apply a model-based engineering process to generate a decision tool what is able to detect from an architecture model which are the feasibility tests that the designer can apply.

1 Introduction

Performance verifications of embedded real-time architectures can be performed with the real-time scheduling theory. Real-time scheduling theory provides analytical methods, called *feasibility tests*, which make possible timing constraints verifications. A lot of feasibility tests have been elaborated during the last 30 years in order to provide a way to compute different performance criteria such as worst case task response time, processor utilization factor and worst case blocking time on shared resources.

Each criterion requires that the target system fulfils a set of specific assumptions that are called *applicability constraints*. Thus, due to the large number of feasibility tests and due to the large number of applicability constraints, it may be difficult for a designer to choose the relevant feasibility test for a given architecture to analyze. Then, it appears that in many practical cases, no such analysis is performed with the help of real-time scheduling theory although experience shows that it could be profitable.

T. Margaria and B. Steffen (Eds.): ISoLA 2010, Part I, LNCS 6415, pp. 4–17, 2010.

In order to help the designer, we have proposed, in [4], a set of architecture design-patterns that allows early performance verifications of architecture models. These design-patterns model usual communication paradigms of multi-tasked real-time software. Given a particular architecture model, these design-patterns are used in order to choose the relevant set of feasibility tests. We have defined four design-patterns called *Synchronous data flows*, *Ravenscar*, *Blackboard* and *Queued buffer*. For each design-pattern, several feasibility tests can be applied. For example, in the case of the *Synchronous data flows* design-pattern, we have listed 10 feasibility tests that can be applied in 64 possible cases, depending on the parameters of each architecture components (tasks, processors, shared resources, etc). It implies that only defining a set of design-patterns may not be enough to really help the designer to automatically perform performance verifications with feasibility tests.

In this article, we investigate how to automatically check that an architecture model is compliant with a design-pattern, in order to ensure that a particular set of feasibility tests is relevant. We show how to explicitly model the relationships between an architectural design-pattern and the compliant feasibility tests. From these models, we apply a model-based engineering process to generate a decision tool which is able to identify, from an architecture model, the feasibility tests the designer is allowed to compute. Then, this decision tool helps the designer to choose the feasibility tests that he is allowed to apply to his architecture models.

This article is organized as follows. In section 2, we introduce our design-pattern approach. Section 3 presents an example: the *Synchronous data flows* design-pattern. In section 4, we explain how the decision tool is currently implemented and how we plan to integrate it into a schedulability tool called *Cheddar*. Then, section 5 is devoted to related works and we conclude and present future works in section 6.

2 The Design-Pattern Approach

During the last decades, a lot of emphasis has been given to software modeling techniques, in a continuous move from traditional coding activities to higher level of abstractions. Several standardized languages such as the MARTE profile for UML [17] or the AADL language [21] provide a set of categorized components that are appropriate for real-time system and software modeling activities. These modeling languages allow not only the applicative architecture to be described, but also its interaction with the underlying executive. As an example, a thread is a kind of AADL component which can be scheduled by the run-time executive.

Nevertheless, although it becomes now easier to describe real-time architectures, their validation still remains a subject of investigation. For instance, the lack of a single property may sometimes be enough to prevent a real-time architecture from being properly processed by a schedulability analysis tool.

This is why, the next step in the improvement of the development process of real-time systems consists in providing to the end user a set of predefined composite constructs that match known real-time scheduling analysis methods.

The composite constructs we have studied correspond to the various inter-task communication paradigms that can be applied in an architecture and that can be considered as real-time design-patterns.

Four design-patterns that are compliant with the real-time scheduling theory are proposed in [4]. These design-patterns are:

1. **Synchronous Data Flows Design-Pattern:** this first design-pattern is the simplest one. Task share data by clock synchronizations: each task reads data at dispatch time or writes data at complete time. This design-pattern does not require the use of shared data components.
2. **Ravenscar Design-Pattern:** the main drawback of the previous pattern is its lack of flexibility at run time. Each task will always execute read and write data at pre-defined times, even if useless. In order to introduce more flexibility, asynchronous inter-task communications is proposed with this design-pattern: tasks access shared data components asynchronously according to priority inheritance protocols.
3. **Blackboard Design-Pattern:** Ravenscar allows task to share data protected by semaphores. Semaphores can be used to build various synchronization protocols such as critical section, barrier, readers-writers, private semaphore, producers-consumers and others [28]. The blackboard design-pattern implements a readers-writers synchronization protocol.
4. **Queued Buffer Design-Pattern:** in the blackboard design-pattern, at any time, only the last written message is made available to the tasks. Queued buffer allows to store all undelivered messages in a memory unit.

Each of these design-patterns models a typical communication and synchronization paradigm of multi-tasked real-time software. Design-patterns are specified with a set of constraints on the architecture model to verify. There are two types of constraints expressed in a design-pattern: *Architectural constraints* and *Property constraints*.

- *Architectural constraints* are restrictive rules which specify the kind of architectural element that are allowed. As an example, a design-pattern can forbid declaration of a shared data component or a buffer in a model. They may also constraint component connections.
- *Property constraints* are related to the architecture components properties and constraint further their value. As an example, a design-pattern can assume that the *quantum*[1] property of a processor component must be equal to zero.

For each design-pattern, according to their architectural and property constraints, we have identified which feasibility tests the designer can compute to perform the verification of his architecture.

With this approach, the designer can verify its real-time system architecture in two steps: (1) he first looks for the design-pattern which is matching his

[1] A quantum is a maximum duration that a task can run on a processor before being preempted by another.

architecture. Then (2), assuming that a matching design-pattern is found, the designer can compute all the feasibility tests associated with this design-pattern.

3 Example of the *Synchronous Data Flows* Design-Pattern

In order to specify our architecture models, we are using the AADL modeling language. AADL is a textual and graphical language for model-based engineering of embedded real-time systems that has been published as SAE Standard AS-5506 [21]. AADL is used to design and analyze software and hardware architectures of embedded real-time systems. Many tools provide support for AADL: Ocarina implements Ada and C code generators for distributed systems [10], TOPCASED, OSATE and Stood provide AADL modeling features [5,3,22], the Fremont toolset and *Cheddar* implement AADL performance analysis methods [26,25]. An updated list of supporting tools can be found on the official AADL web site http://www.aadl.info.

An AADL model describes both the hardware part and the software part of an embedded real-time system. Basically, an AADL model is composed of components with different categories: data, threads or processes (components modeling the software side of a specification), processors, devices and buses (components modeling the hardware side of a specification). A data component may represent a data structure in the program source text. It may contain subprograms such as functions or procedures. A thread is a sequential flow of control that executes a program and can be implemented by an Ada task or a POSIX thread. AADL threads can be dispatched according to several policies: a thread may be periodic, sporadic or aperiodic. An AADL process models an address space. In the most simple case, a process contains threads and data. Finally, processors, buses and devices represent hardware components running one or several applications.

This section presents one of the simplest design-patterns: the *Synchronous data flows* design-pattern. First, we specify the design-pattern by its architectural and property constraints. Then, we present the feasibility tests that are assigned to this design-pattern. In the sequel, we also illustrate this design-pattern with a compliant AADL model.

3.1 Specification of the *Synchronous Data Flows* Design-Pattern

The *Synchronous data flows* design-pattern is inherited from *Meta-H*. An AADL architecture model is compliant with this design-pattern if it is only composed of process, thread, sub-program and processor components. We also assume that the architecture model meets the constraints expressed by the following seven rules:

Rule 1: All threads are periodic.

Rule 2: We assume that threads are scheduled either by a fixed priority scheduler or by EDF [14]. In the case of a fixed priority scheduler, any kind of priority assignment can be used (*Rate Monotonic* or *Deadline Monotonic*) but we assume that all threads have different priority levels.

Rule 3: The scheduler may be either fully preemptive or non preemptive.

Rule 4: We assume that the scheduler do not use *quantum* (see the POSIX 1003 scheduling model [7]).

Rule 5: Thread communications do not make use of any data component, of any shared resource or buffer and there is no connection between threads and data components.

Rule 6: Threads are independent: thread dispatches are not affected by the inter-thread communications. In this synchronization schema, communications between threads are achieved by pure data flows with AADL data ports: each thread reads input data at dispatch time and writes output data at completion time.

Rule 7: Each processor owns only one process and there is no virtual processor. This rule expresses that there is no hierarchical scheduling (see the ARINC 653 standard [1]).

3.2 Feasibility Tests Assigned to the *Synchronous Data Flows* Design-Pattern

For an AADL model compliant with *Synchronous data flows*, we can perform performance analysis with real-time scheduling theory feasibility tests. For this design-pattern, we can check performances by computing two performance criteria: (1) the worst case response time of each thread and (2) the processor utilization factor. For such a purpose, we have assigned 10 feasibility tests to compute these performance criteria [23].

However, it does not mean that for each AADL model compliant with this design-pattern, we can apply the 10 feasibility tests. For a given AADL model, depending on the value of the AADL component properties, we will be able to apply one or several feasibility tests among this set of feasibility tests. We have identified 64 different cases depending on component properties values, which represent applicability assumptions of the feasibility tests. This shows that even for this simplest design-pattern, choosing the right feasibility test to apply may be difficult for architecture designers.

One of these feasibility tests is called the "worst case response time feasibility test" and consists in comparing the worst case response time of each thread with its deadline.

For this feasibility test, the thread components of the *Synchronous data flows* design-pattern are defined by three parameters: their deadline (D_i), their period (P_i) and their capacity (C_i). P_i is a fixed delay between two release times of the thread i. Each time the thread i is released, it has to do a job whose execution time is bounded by C_i units of time. This job has to be ended before D_i units of time after the thread release time.

Joseph and Pandia [13] have proposed a way to compute the worst case response time of a thread with pre-emptive fixed priority scheduling by equation:

$$r_i = C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{r_i}{P_j} \right\rceil \cdot C_j \qquad (1)$$

Where r_i is the worst case response time of thread i and $hp(i)$ is the set of threads that have a higher priority level than thread i.

This feasibility test is one of the most simple tests which can be applied to the *Synchronous data flows* design-pattern. This test has several applicability assumptions. For example, this test assumes that deadlines are equal to periods and that all threads have an equal first release time.

3.3 Example of an AADL Model Compliant with the *Synchronous Data Flows* Design-Pattern

Figure 1 shows a part of an AADL model. This example is composed of several periodic threads defined into a process and that are run on a single processor. AADL component are always specified by a type definition (line number 1) and a component implementation (from line 2 to line 9). The model also contains several AADL properties. Some properties of thread components define deadlines, periods, capacities and the thread priority. The properties of the processor component defines how the scheduler works. In this example, the processor embeds a preemptive Rate Monotonic scheduler with a quantum equal to zero (from line 24 to line 27).

This AADL model is compliant with the *Synchronous data flows* design-pattern because all rules of the section 3.1 are met. We assume that the designer has previously checked the compliance of this model with this set of rules.

```
thread T1 end T1;                                                          1
                                                                           2
thread implementation T1.impl                                              3
properties                                                                 4
    Dispatch_Protocol => Periodic;                                         5
    Compute_Execution_time => 1 ms .. 2 ms;                                6
    Deadline => 10 ms;                                                     7
    Period => 10 ms;                                                       8
    Cheddar_Properties::Fixed_Priority => 128;                             9
end T1.impl;                                                               10
                                                                           11
thread T2 end T2;                                                          12
                                                                           13
thread implementation T2.impl                                              14
properties ....                                                            15
end T2.impl                                                                16
                                                                           17
process implementation process0.impl                                      18
subcomponents                                                              19
  a_T2 : thread T2.impl; ....                                              20
end process0.impl                                                          21
                                                                           22
processor implementation rma_cpu.impl                                      23
properties                                                                 24
   Scheduling_Protocol => Rate_Monotonic_Protocol;                         25
   Cheddar_Properties::Preemptive_Scheduler => True;                       26
   Cheddar_Properties::Scheduler_Quantum => 0 ms;                          27
end rma_cpu.impl;                                                          28
```

Fig. 1. Part of an AADL model

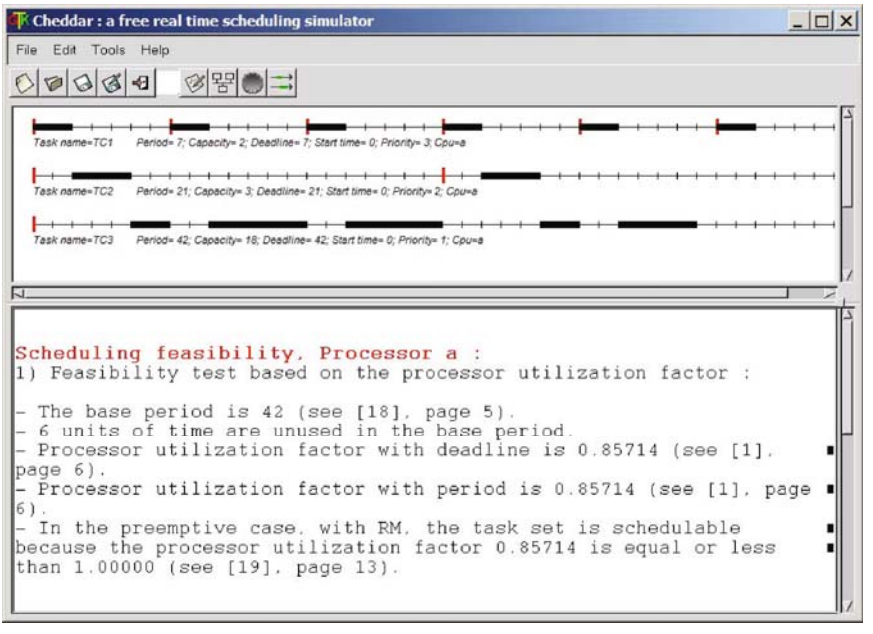


Fig. 2. A screenshot of Cheddar, a tool which implements several feasibility tests

To compute feasibility tests of the *Synchronous data flows* design-pattern, we are using *Cheddar*. *Cheddar* is a framework which aims at providing performance analysis of concurrent real-time applications [25]. With *Cheddar*, a real-time application is modeled as a set of processors, shared resources, buffers and tasks. *Cheddar* is able to handle architecture models described by an AADL specification and also by its own simplified architecture design language. *Cheddar* already implements numerous feasibility tests [23]. Figure 2 shows a screenshot of *Cheddar*. The top part of this window displays the scheduling analysis of the model of figure 1 and the bottom part shows the feasibility tests results computed for this AADL model.

4 A Decision Tool to Check the Compliance of an AADL Model with the Design-Patterns

In the previous section, we have presented a design-pattern and an AADL model that is compliant with it. To automatically perform performance verifications of an AADL model with *Cheddar*, we have assumed that designers are able to check the compliance of their AADL models with a design-pattern. This task is, however, not easy for users who are not experts on real-time scheduling theory. To facilitate this task, we propose a second tool, called *decision tool*, which is able to automatically perform this compliance analysis.

We plan to integrate this decision tool as a new functionality of *Cheddar*. As depicted by figure 3, within *Cheddar*, the decision tool will be able to automatically

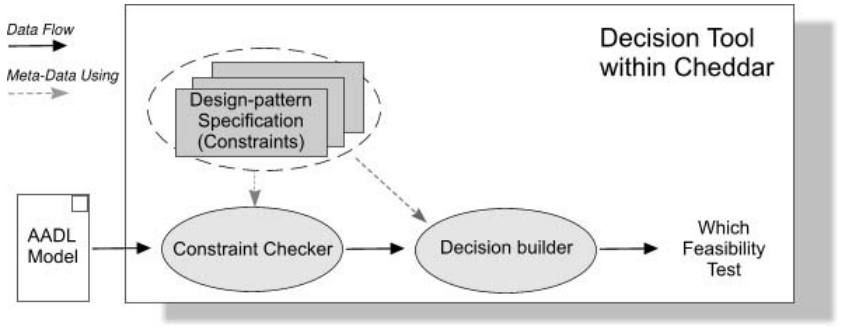


Fig. 3. AADL model analyzer overview

detect which design-pattern a real-time system is compliant with and then to automatically compute the relevant feasibility tests.

So far, the design-patterns are themselves currently elaborated. Thus, for now we are using a prototype of the decision tool. This prototype is built with the *Platypus* tools. In this section we first briefly describes the *Platypus* tool and its usage for prototyping the decision tool. Then we explain how *Cheddar* will be enriched with the decision tool.

4.1 Prototyping within Platypus

Platypus [20] is a software engineering tool which embeds a modeling environment based on the STEP standard [11]. First of all, *Platypus* is a STEP environment, allowing data modeling with the EXPRESS language [12] and the implementation of STEP exchange components automatically generated from EXPRESS models. *Platypus* includes an EXPRESS editor and checker as well as a STEP file reader, writer and checker.

In *Platypus*, a meta-model consists in a set of *EXPRESS* schemas that can be used to describe a language. The main components of the meta-model are types and entities.

From an EXPRESS schema and a data set made of instances of entities described by the EXPRESS schema, *Platypus* is able to check the data set conformity by evaluating the constraint rules specified in the EXPRESS model.

Thus, given that the design-patterns are specified with EXPRESS, the decision tool prototype directly benefits from the *Platypus* STEP generic framework. The figure 4 shows the prototype components and the data flow when an architecture model is analyzed. The prototype is first made of the shared meta-model named *Cheddar meta-model schema*. This meta-model specifies the internal *Cheddar* representation of a real-time architecture to verify. Then, each design-pattern is composed of two models which are defined in order to further constraint the *Cheddar* meta-model. These models correspond to the two kinds of constraints as explained in the section 2. In the figure 4, they are specified by the *architectural constraints* and *property constraints* schemas.

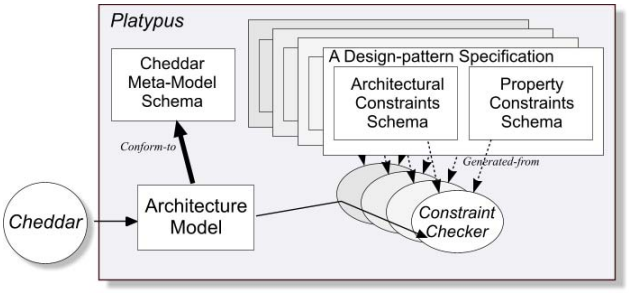


Fig. 4. The decision tool prototype within Platypus

Due to the current usage of the prototype, in order to be analyzed, an AADL architecture model must be encoded as an XML or a STEP data exchange file conforming to the *Cheddar meta-model*. *Cheddar* can be used for that purpose. Then, each design-pattern is evaluated separately. Evaluating a design-pattern consists in interpreting all rules specified in the *architectural constraints* and *property constraints* schemas.

4.2 Design-Pattern Modeling Framework

As depicted by figure 5, the modeling framework is made of three main layers: (1) the *Architecture resources*, (2) the *Feasibility test resources* and (3) the *Feasibility test design-patterns* layers. Each of these layers are made of one or several EXPRESS schemas. This section briefly describes them and gives some illustrative EXPRESS samples.

The *Architecture resources* layer. This layer is the most generic one, it contains the specification of all domain entities which are used for architectures

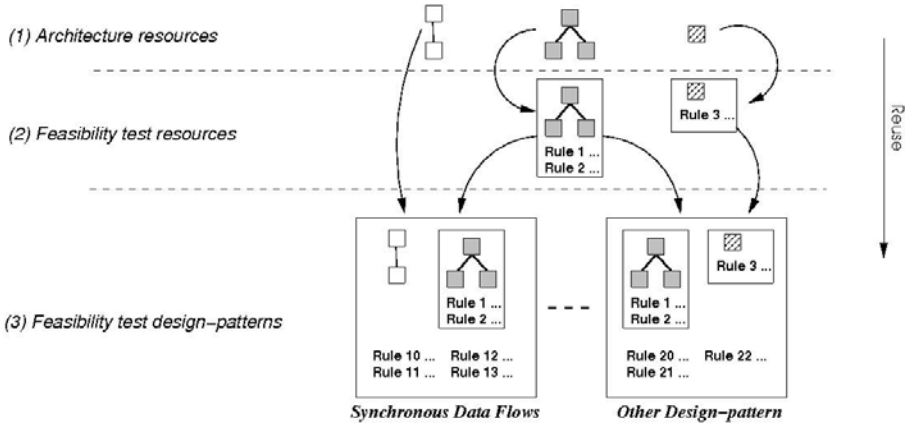


Fig. 5. The three layers of the Design-pattern modeling framework

```
SCHEMA Tasks;
    ENTITY Periodic_Task SUBTYPE OF ( Generic_Task );
        Period : Natural_type;
        Jitter : Natural_type;
    END_ENTITY; ...
END_SCHEMA;

SCHEMA Schedulers;
    TYPE Preemptive_Type = ENUMERATION OF
        (fully_preemptive, non_preemptive, partially_preemptive);
    END_TYPE;

    ENTITY Generic_Scheduler;
        Quantum : Natural_type;
        Preemptivity : Preemptive_Type;
    END_ENTITY; ...
END_SCHEMA;

SCHEMA Processors ... END_SCHEMA;
SCHEMA Buffers ... END_SCHEMA;
```

Fig. 6. Part the Cheddar Meta-model

modeling. Indeed, it is specified independently of any design-pattern. It mainly contains the Cheddar meta-model. A particular real-time architecture is made of instances of this meta-model because it specifies the internal representation of a real-time architecture within Cheddar. As depicted by the figure 6, it is composed of entities such as *Generic_Task* or *Generic_Scheduler* but also *Buffer* or *Processor*. These entities store all attributes that are required for the analysis of an AADL architecture model.

The *Feasibility test resources* layer. This is the intermediate layer which makes use of entities of the *Architecture resources* layer and in addition is made of new entities, functions and rules which are reusable across several design-patterns.

```
SCHEMA Period_Equal_Deadline_Constraint;
    USE FROM Tasks;

    RULE Period_Equal_Deadline FOR ( Periodic_Task );
    WHERE
        SIZEOF ( QUERY ( p <* Periodic_Task | p.Period <> p.Deadline ) ) = 0;
    END_RULE;
END_SCHEMA;
```

The *Period_Equal_Deadline* constraint concerns only the set of all *Periodic_Task* instances. This constraint is satisfied if there is no *Periodic_Task* instance which have a period value which is different from its deadline.

Fig. 7. EXPRESS model of the Feasibility tests resources layer

As an example, the *Period_Equal_Deadline* constraint is shown in the figure 7. This constraint is typically reusable for several design-patterns. It is specified by an EXPRESS rule which ensure that, for each instance of the *Periodic_Task* entity, the value of the *Period* and of the *Deadline* attributes are equal.

The *Feasibility test design-patterns* layer. Each model of this layer is called a *design-pattern model*, it concerns a particular design-pattern and is made of one or several EXPRESS schemas which reuse parts of the other layers. In addition, each *design-pattern model* defines very specific rules that represent constraints which are to be checked for the related design-pattern.

```
SCHEMA Data_Flow_Constraints;                                                    57
    USE FROM Tasks;  USE FROM Schedulers;  USE FROM Buffers;                     58
                                                                                 59
    RULE All_Tasks_Are_Periodic FOR ( Generic_Task );                            60
    WHERE                                                                        61
        R1 : SIZEOF ( QUERY ( t <* Generic_Task |                                62
             NOT ('TASKS.PERIODIC_TASK' IN TYPEOF ( t ) ) ) ) = 0;               63
    END_RULE;                                                                    64
    ...                                                                          65
    RULE No_Shared_Ressource FOR ( Generic_Resource, Buffer );                   66
    WHERE                                                                        67
        R5 : ( SIZEOF ( generic_resource ) = 0 ) AND ( SIZEOF ( Buffer ) = 0 );  68
    END_RULE;                                                                    69
END_SCHEMA;                                                                      70
```

Fig. 8. EXPRESS model for the data flow design-pattern

As an example, the *Data_Flow_Constraints* EXPRESS schema shown in figure 8 is for the *Data Flow* design-pattern. Only the rules 1 and 5 given for this design-pattern are shown (see section 3.1 page 7): the *Rule 1* is specified by the *All_Tasks_Are_Periodic* EXPRESS constraint and the *Rule 5* by the *No_Shared_Ressource* one.

4.3 Toward an Implementation within Cheddar

As we are currently elaborating EXPRESS models for the work presented in this article, it is very efficient to be able to directly test them from the *Platypus* modeling environment itself.

But having to use *Cheddar* together with *Platypus* is not comfortable for end-users and *Platypus*, as a STEP based data modeling environment, is not user friendly enough. We plan to use a model driven engineering process in order to automatically generate the decision tool in Ada, the implementation language of *Cheddar*. For such a purpose, *Platypys* will have to handle EXPRESS models of figures 6, 7 and 8. Today, *Cheddar* is already partly automatically generated by *Platypus* from EXPRESS models of the figure 6. As an example, the *Cheddar meta-model* schema is used in order to produce the core components of *Cheddar* [19,24].

5 Related Works

This article has shown an approach to check that an architectural model of a real-time system is compliant with a set of constraints. Many other approaches also investigated how to perform such verifications.

First, UML together with its standard constraint language OCL could be used for the purpose of designing and building feasibility test checkers.

Second, in [8], Gilles and al. have proposed a similar constraint language for AADL. The proposed language is called REAL (REAL stands for Requirement Enforcement Analysis Language). REAL is developed by Télécom-Paris-Tech and ISAE. It should be adopted as an annex of the AADL standard. This language is then specifically designed for the modeling of real-time architectures. REAL allows to express various type of constraints on AADL architecture and their authors have shown that it can express some of the applicability constraints of the real-time scheduling theory.

Another approach of a similar move towards more analyzable constructs built on top of a modeling language can be found in the history of the HOOD method [15]. The first versions of this modeling approach defined a quite basic concept of component (called HOOD objects) which aimed at representing more or less an Ada 83 package. In 1995, two specializations of HOOD were specified: HOOD 4 [16] which targets Object-Oriented programming languages and especially Ada 95, and HRT-HOOD [2] which goal is to comply with the Ada Ravenscar model (now included into Ada 2005 [27]). In both cases, the original concepts and principles of the HOOD methodology have been kept, and specific composite constructs have been identified in order to support properly Ada 95 tagged types or Ravenscar cyclic, sporadic and protected objects. More recently, in the context of the IST-ASSERT project, Panunzio and al. [18] proposed to integrate some HRT-HOOD components with UML models. For such a purpose, they have proposed an engineering process based on a meta-model called RCM (RCM stands for Ravenscar Computational Model). In this process, performance verifications are performed with the MAST framework [9], which also implements several feasibility tests.

Finally, PPOOA proposes a similar approach [6]. PPOOA is an architectural style for concurrent object oriented architectures. PPOOA is implemented as an extension of UML and provide several coordination mechanisms such as buffers, semaphores, transporters, Ada rendezvous and others. All these coordination mechanisms are similar to our design-patterns.

6 Conclusion

Feasibility tests of real-time scheduling theory may be difficult to be used by system designers. In this article, we investigate how to increase their usability with an approach based on design-patterns.

In this approach, we have defined a list of design-patterns and a set of feasibility tests is assigned to each design-pattern. When a designer wants to perform a performance analysis of an AADL model, he must check that his model is compliant with one of these design-patterns. If the model is actually compliant with a design-pattern then he can call *Cheddar* to automatically compute the feasibility tests assigned to the selected design-pattern.

However, checking compliance of his models to the design-patterns may be difficult to achieve, especially if designers are not expert on real-time scheduling theory.

To automatically check compliance, we propose a framework, called decision tool, which relies on the Platypus environment.

In the current implementation of our approach, designers have to handle two different tools: *Cheddar* and the decision tool. In a next step, we plan to integrate the decision tool into *Cheddar*. Having only one tool to deal with should facilitate the performance analysis of AADL models.

A second future work is related to the list of design-patterns. Indeed, we only have investigated how to check compliance with the *Synchronous data flows* design-pattern. In the next months, we will do the same work for the other design-patterns: *Ravenscar*, *Queued Buffer* and *BlackBoard* [4].

Finally, in this approach, we expect to verify if an AADL model is fully compliant with a set of design-patterns. But in some cases, architectural models of practitioners may be compliant with none of the proposed design-patterns. Then, we plan to investigate how the designers can eventually be helped with a set of metrics. These metrics should allow the designers to compare their AADL models with our design-patterns and to improve their models in order to be compliant with the real-time scheduling theory.

References

1. Arinc: Avionics Application Software Standard Interface. The Arinc Committee (January 1997)
2. Burns, A., Wellings, A.: HRT-HOOD: A Design Method for Hard Real-time Systems. Real Time Systems Journal 6(1), 73–114 (1994)
3. Dissaux, P.: Using the AADL for mission critical software development. In: 2nd European Congress Erts, Embedded Real Time Software Toulouse (January 2004)
4. Dissaux, P., Singhoff, F.: Stood and Cheddar: AADL as a Pivot Language for Analysing Performances of Real Time Architectures. In: Proceedings of the European Real Time System Conference, Toulouse, France (January 2008)
5. Farail, P., Gaufillet, P., Canals, A., Camus, C.L., Sciamma, D., Michel, P., Crégut, X., Pantel, M.: From MDD Concepts to Experiments and Illustrations. In: ISTE (ed.) TOPCASED: An Open Source Development Environment for Embedded Systems, ch. 11, pp. 195–207 (September 2006)
6. Fernandez, J.L., Marmol, G.: An Effective Collaboration of a Modeling Tool and a Simulation and Evaluation Framework. In: 18th Annual International Symposium, INCOSE 2008, Systems Engineering for the Planet, The Netherlands (June 2008)
7. Gallmeister, B.O.: POSIX 4: Programming for the Real World. O'Reilly and Associates, Sebastopol (January 1995)
8. Gilles, O., Hugues, J.: Expressing and enforcing user-defined constraints of AADL models. In: The Proceedings of the 15th IEEE International Conference on Engineering of Complex Computer Systems, International workshop on AADL and UML, University of Oxford, UK, pp. 337–348 (March 2010)
9. Harbour, M.G., Garcìa, J.G., Gutiérrez, J.P., Moyano, J.D.: MAST: Modeling and Analysis Suite for Real Time Applications. In: Proc. of the 13th Euromicro Conference on Real-Time Systems, Delft, The Netherlands, pp. 125–134 (June 2001)
10. Hugues, J., Zalila, B., Pautet, L., Kordon, F.: Rapid Prototyping of Distributed Real-Time Embedded Systems Using the AADL and Ocarina. In: 18th IEEE/IFIP

International Workshop on Rapid System Prototyping (RSP 2007), Porto Allegre, Brazil (June 2007)
11. ISO 10303-1: Part 1: Overview and fundamental principles (1994)
12. ISO 10303-11: Part 11: edition 2, EXPRESS Language Reference Manual (2004)
13. Joseph, M., Pandya, P.: Finding Response Time in a Real-Time System. Computer Journal 29(5), 390–395 (1986)
14. Liu, C.L., Layland, J.W.: Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environnment. Journal of the Association for Computing Machinery 20(1), 46–61 (1973)
15. Masson and Prentice-Hall: HOOD Reference Manual release 3.1, HOOD User Group (1993)
16. Masson and Prentice-Hall: HOOD Reference Manual release 4.0, HOOD User Group (1995)
17. OMG: A UML Profile for MARTE, Beta 1. OMG Document Number: ptc/07-08-04 (August 2007)
18. Panunzio, M., Vardanega, T.: A Metamodel-Driven Process Featuring Advanced Model-Based Timing Analysis. In: Abdennahder, N., Kordon, F. (eds.) Ada-Europe 2007. LNCS, vol. 4498, pp. 128–141. Springer, Heidelberg (2007)
19. Plantec, A., Singhoff, F.: Refactoring of an Ada 95 Library with a Meta CASE Tool. ACM SIGAda Ada Letters 26(3), 61–70 (2006)
20. Platypus Technical Summary and download (2007), http://cassoulet.univ-brest.fr/mme/
21. SAE: Architecture Analysis and Design Language (AADL) AS 5506. Tech. rep., The Engineering Society For Advancing Mobility Land Sea Air and Space, Aerospace Information Report, Version 2.0 (January 2009)
22. SEI: OSATE : An extensible Source AADL Tool Environment. SEI AADL Team technical Report (December 2004)
23. Singhoff, F.: A taxonomy of real-time scheduling theory feasibility tests. LISyC Technical report, number singhoff-01-2010 (February 2010), http://beru.univ-brest.fr/~singhoff/cheddar
24. Singhoff, F., Plantec, A.: Towards User-Level extensibility of an Ada library: an experiment with Cheddar. In: Abdennahder, N., Kordon, F. (eds.) Ada-Europe 2007. LNCS, vol. 4498, pp. 180–191. Springer, Heidelberg (2007)
25. Singhoff, F., Plantec, A., Dissaux, P., Legrand, J.: Investigating the usability of real-time scheduling theory with the Cheddar project. Journal of Real-Time Systems 43(3), 259–295 (2009)
26. Sokolsky, O., Lee, I., Clark, D.: Schedulability Analysis of AADL models. In: International Parallel and Distributed Processing Symposium, IPDPS 2006, vol. 2006 (April 2006)
27. Taft, S.T., Duff, R.A., Brukardt, R.L., Ploedereder, E., Leroy, P.: Ada 2005 Reference Manual. Language and Standard Libraries. International Standard ISO/IEC 8652/1995(E) with Technical Corrigendum 1 and Amendment 1. LNCS, vol. 4348. Springer, Heidelberg (2006)
28. Tanenbaum, A.: Modern Operating Systems. Prentice-Hall, Englewood Cliffs (2001)

Seamless Model-Driven Development Put into Practice

Wolfgang Haberl[1], Markus Herrmannsdoerfer[1], Stefan Kugele[1], Michael Tautschnig[2], and Martin Wechs[3]

[1] Institut für Informatik
Technische Universität München
Boltzmannstr. 3, 85748 Garching b. München, Germany
[2] Formal Methods in Systems Engineering, Faculty of Informatics
Vienna University of Technology
Favoritenstr. 9, 1040 Wien, Austria
[3] BMW Forschung und Technik GmbH
Hanauer Straße 46, 80992 München, Germany

Abstract. Model-driven development (MDD) today is the most promising approach to handle the complexity of software development for distributed embedded systems. Still, no single tool-chain exists that meets all needs of companies employing MDD. Moving back and forth between the tools in today's iterative development processes thus requires manual integration steps, which are error-prone and hamper reuse and refinement of models. A possible workaround is developing adapters for each pair of tools. Despite its large overhead, industry started pursuing this approach because of a lack of better alternatives. A proper solution is a tool-chain building on an *integrated* modeling language. We have realized this in cooperation with BMW Research and Technology. To increase the degree of automation during development, the modeling language builds upon a core featuring a rigorous semantics. This enables automatic analysis, facilitating an automatic transition from model-based designs to a distributed system running on the target platform.

1 Introduction

As embedded systems, ranging from music players and mobile phones over cardiac pacemakers to airbag controllers, become ubiquitous, the need for quality escalates. Up to 98% of all processors are built into embedded systems [1,2]. Many of these perform mission-critical tasks in systems such as flight controllers or air condition systems in premium-class cars. In such large-scale safety-critical systems, an imminent need for quality and absence of errors meets the complexity of developing software for distributed systems.

In industry, model-driven development (MDD) is seen as the most promising technique to tackle this complexity. Consequently, software industry created a wide range of tools to aid the developer in many steps of an MDD process. Still, there is not a seamless tool-chain, let alone an integrated tool, covering the entire

T. Margaria and B. Steffen (Eds.): ISoLA 2010, Part I, LNCS 6415, pp. 18–32, 2010.

development process from requirements and design models to code executable on the target platform. Instead, developers must deal with a multitude of tools, which requires converting process artifacts back and forth as the product evolves.

Industry tries to work around these largely manual steps by building tool adapters. This approach may seem to be the simplest at first sight, but it is surely not future-proof. This is, however, the least of a problem: in fact, it is often technically unsound. An automated translation between tools requires that the underlying modeling languages have well-defined and compatible semantics. As this is often not the case, each transition—be it manual or automatic—bears the risks of loss of information and inadvertent introduction of additional errors.

Looking at the amount and structure of warranty costs in the automotive domain, a need for improvement is imminent: billions of dollars are spent for warranty costs each year. These costs amount to a total of up to US$ 500 per vehicle [3]. According to IBM research, about 30% of these costs are attributable to software and electronics defects. With current tools, a decrease in the number of implementation errors seems largely out of reach. In industrial embedded systems development, however, not only the number of design and implementation errors poses a challenge. Architectural and cost constraints require optimization at system level. Due to the isolation of tools, currently optimization can only be applied locally. In consequence, automatic global optimizations—such as fewer and cheaper controllers, smaller packaging, or reduced weight—are impossible.

As the amount of software in cars continues to grow exponentially, improved tools alone will not suffice. Instead, an appropriate development process *and* corresponding tool support will be required. Especially in the automotive domain, struggling with the enormous cost pressure, this is an important issue. Short product cycles require that iterative processes have both short and few iterations, requiring best possible automation at low overhead. All these facts collide with the state-of-the-art, where manual and error prone conversion is involved. In cooperation with BMW Research and Technology, we developed an approach to overcome those hurdles. In this paper we give an overview of our process and tool chain. We refer to previous publications describing specific technical aspects in detail where appropriate.

2 Requirements for Seamless Model-Driven Development

To tackle these, we propose a seamless model-driven development approach [4]. Starting from requirements and continuing to design and code generation, deep integration of all constituents is required. More specifically, this is required for

- the *modeling language* which provides an easy-to-use syntax and a well-defined semantics for all artifacts,
- the *modeling process* which describes how to create the artifacts step by step using the modeling language, and
- the *modeling tool* which supports the developer to author the artifacts and automates the process steps as far as possible.

In the following, we will detail on the necessary prerequisites to provide integration for each of these constituents.

Integrated Modeling Language. An integrated modeling language enables integration of all artifacts created during the development process. The integrated modeling language shall rigorously define the syntactic structure of all artifacts as well as their relation. Moreover, a concrete syntax that visualizes the artifacts in a human-understandable way has to be provided by the integrated modeling language. It needs to be based on a common modeling theory giving the artifacts a precise semantics. First, this theory is essential to prove properties over all kinds of artifacts created along different process steps. Second, model transformation as well as code generation require a common modeling theory to ensure preservation of semantics when advancing between process steps. The integrated modeling language must support all process steps in a seamless manner to enable an integrated modeling process.

Integrated Modeling Process. An integrated modeling process is required to seamlessly develop an embedded system in a stepwise manner. The integrated modeling process needs to define the interplay of all the process steps necessary to create the desired artifacts. Each process step needs to be defined in a way such that it can be either automated by a model transformation or at least supported by the modeling tool. The integrated modeling process must be able to cope with the stringent constraints of embedded systems on reliability, robustness, correctness, and an overall optimized system with respect to execution time and resource usage. Building on the well-defined semantics of the modeling language, verification is performed continuously together with design and implementation activities. Consequently, many design errors can already be ruled out at higher levels of abstraction, reducing cost induced by correcting these errors at a later stage.

Integrated Modeling Tool. An integrated modeling tool is required to provide seamless tool support for the development of all artifacts. It needs to be based on a central repository that contains all artifacts and their relationships. The central repository avoids redundancy and inconsistency that results from parallel modification of artifacts by different developers. All the authoring and transformation tools provided by the integrated modeling tool need to operate directly on this central repository—supervised by a well-defined concurrency control. Moreover, seamless change and configuration management is only possible by maintaining all the artifacts in the central repository. The integrated modeling tool shall further enable pervasive tool support to guide the engineers throughout the modeling process.

3 Realization of Seamless Model-Driven Development

Following the requirements listed in the preceding section, we describe a solution to the problems stated in Section 1. We first give a short account of the integrated modeling language COLA, and then outline its use in both the integrated modeling process and the integrated modeling tool.

3.1 COLA – The Component Language

COLA, the COmponent LAnguage, is an integrated modeling language for the design of distributed, safety-critical real-time systems. The key concepts of COLA follow those of well-known modeling languages, such as FODA [5] and Lustre [6]. The resulting language is structured into different layers of abstraction along which a system is modeled. These layers help to reduce the complexity involved in the design of large-scale embedded systems. Each abstraction layer focuses on a certain aspect, and adds new detail to the information contained in higher layers. COLA defines three layers of abstraction which are called Feature Architecture (FA), Logical Architecture (LA), and Technical Architecture (TA), as proposed by Pretschner et al. [7] and Broy et al. [8]. Figure 1 illustrates the core modeling constructs of each layer, and the relations between them, in screenshots from our integrated modeling tool.

COLA is based on the synchronous data-flow paradigm and has a well-defined semantics [9]. It is assumed that operations start at the same instant of time and are performed simultaneously, respecting data dependencies. The computation of the system over time can be subdivided into discrete steps, called *ticks*. The execution is performed in a stepwise manner over a discrete uniform time-base.

Feature Architecture (FA). The *Feature Architecture* formalizes functional requirements on behavior observable at the system boundary [10]. Similar to FODA [5], the overall system behavior is decomposed into a hierarchy of *features* to reduce complexity. A feature is a function which is observable by the customer at the system boundary. The inner nodes of the tree are *composite features* which are decomposed into sub-features. The leaves of the tree represent *atomic features* which cannot be decomposed further and model a function accessible to the customer. Figure 1 depicts the graphical representation of an example feature hierarchy. In the graphical syntax of COLA, the feature hierarchy is represented as a tree, the sub-feature relationship as lines with a diamond, and features as rounded rectangles. In a complex system, however, features are not independent, but rather influence each other. These dependencies are represented as arrows between the corresponding features (see Figure 1). We extended FODA to be able to rigorously define the behavior of atomic and composite features as well as feature dependencies. To ease the transition to the next layer, the behavior is defined by means of constructs provided by the LA, e. g., data-flow networks.

Logical Architecture (LA). The *Logical Architecture* maps features onto a software architecture, and refines them to obtain a platform-independent model of the overall system functionality. The constructs to model the system functionality are based on Lustre [6]. The *system* can be successively decomposed into or composed of modular *units*. These units may be defined in-place, as described next, or taken from user-defined libraries, which enable reuse. The behavior of a unit is hidden behind the unit's interface which is defined by a number of typed input and output *ports*. Figure 1 depicts the graphical representation of a number of units: interfaces are represented as rounded rectangles and ports as

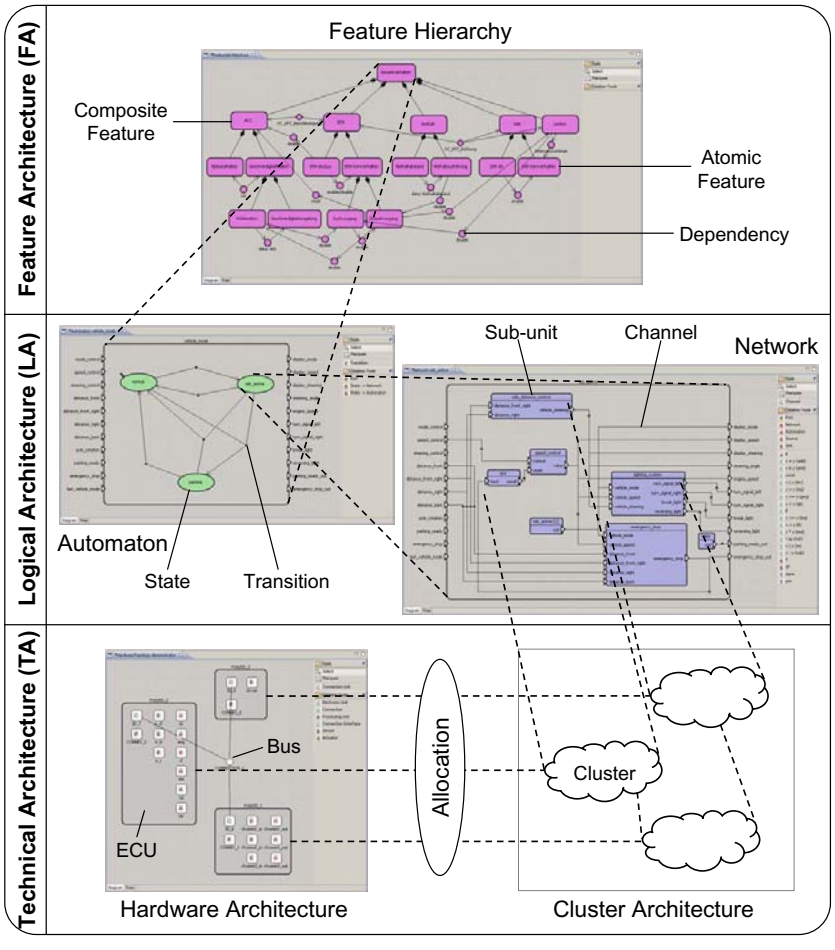


Fig. 1. Integrated Modeling Language COLA

triangles. The root unit of a system does not have any unconnected input or output ports, but communicates with the environment using so-called *sources* and *sinks*. These sources and sinks model sensor and actuator interaction. COLA defines three kinds of units: networks, automata, and blocks.

A system can be decomposed into smaller units, establishing data-flow *networks*. These smaller units are called *sub-units* of the network, which are connected by *channels*. A channel connects an output port with one or more suitably typed (cf. [11]) input ports. The synchronous semantics of COLA define communication over channels to take no time. Channels determine data-flow dependencies between sub-units and induce a causal order of execution. Feedback loops of channels (transitively) connecting an output port of a unit to an input port of the same unit must contain a *delay* block which defers propagation by one time interval. A delay serves as data storage: it retains values from one tick to the next. Figure 1 depicts the graphical representation of an example

network: sub-units are represented by rounded rectangles, and channels by lines connecting the sub-units.

Control flow in networks is modeled by *automata*. An automaton consists of a number of states where exactly one is active at a time. The automaton's behavior is defined by the currently active state. Each state is implemented by a sub-unit. *Transitions* determine how the activated state of an automaton may change over time. The condition for a transition is again implemented by a unit, the evaluation of which is based on the inputs of the automaton. The semantics of COLA requires to check for possible transitions before evaluating the activated state's behavior. Figure 1 depicts the graphical representation of an example automaton: states are represented by ellipses, and transitions by arrows.

COLA provides a number of basic building *blocks*, e. g., arithmetic or Boolean operators. These blocks execute their respective operation based on the values present at the input ports and emit the according result at their output port.

In the course of computation, a unit may act differently depending on its history. Such units are considered *stateful*. In COLA, only delays and automata retain information of previous computations; all other kinds of units are *stateless*. Note that a unit containing a stateful sub-unit becomes stateful as well.

Technical Architecture (TA). The *Technical Architecture* maps system functionality, specified by the LA, onto a hardware platform. As shown in Figure 1, the TA consists of Hardware Architecture, Cluster Architecture, and Allocation.

The *Hardware Architecture* describes the structure and the properties of the target hardware platform. The main construct of the Hardware Architecture is the *electronic control unit* (ECU) which forms a computing node of the distributed platform and can be composed of *processors*, *sensors*, and *actuators*. Sensors and actuators model interaction with the environment, whereas processors execute the system functionality. ECUs can be connected via a *bus* which provides a communication mechanism. The characteristics of each hardware element, e. g., the resources provided by a processor, can be determined by a number of properties, e. g., clock frequency, storage capacity, etc.

The *Cluster Architecture* partitions the system functionality—as defined in the LA—into distributable entities, called *clusters*. These will be mapped to the electronic control units of the Hardware Architecture. The Cluster Architecture may be specified manually by the developer, or can be derived automatically from the LA based on characteristics specified in the Hardware Architecture, such as available processing speed or memory. In the latter case, the derivation of the Cluster Architecture can be performed based on some optimization criteria, like for example shortest average turnaround time for all tasks on all nodes, or equally distributed memory consumption for all nodes.

The *Allocation* establishes the relationship between the Cluster Architecture and the Hardware Architecture. Therefore the Allocation maps each cluster onto an ECU. For the Allocation to be valid, the resources provided by each ECU, like computing power, memory, etc., have to match the resource requirements of the clusters placed thereon for all nodes of the system. To estimate the resource requirements of a cluster, its worst-case execution time (WCET) needs to be

calculated. The Allocation can then be automatically generated based on both Hardware and Cluster Architecture taking into account optimization criteria.

3.2 Model Analysis and System Synthesis

The automated transition from a modeled system to executable code is supported by our integrated modeling tool. This transition directly follows the proposed process. We will detail on the necessary steps below, due to space limitations focusing on the transformation from the LA down to an executable system. The artifacts arising during the described transition are depicted in Figure 2.

Model Analysis. The formal semantics of our modeling language enables automatic analysis. For such a technique to be useful for the systems engineer, it must not only report the presence of errors, but also yield diagnostic information, i. e., provide the reason of a problem. Furthermore, we require all analyses to be push-button. We have implemented a stack of such methods in our tool.

First, consistency checks according to constraints, which are part of our formal modeling language, are performed. This analysis reports syntax errors and violated model invariants. On top of our modeling language we provide a static type system with type variables, which includes an extension to support physical units. Automatic type inference detects and diagnoses errors at interconnected component interfaces, i. e., syntactically incompatible interfaces [11].

To complement these syntactic checks, COLA enables several static semantic analyses: we have implemented a check to verify the absence of nondeterminism in the modeled automata. This analysis reports states of automata that may have more than one activated outgoing transition at the same time. Such behavior must be avoided as code generated from the model would otherwise define the order of execution in a non-foreseeable manner. Further we interface with model checkers to verify the conformance of a design to requirements expressed in SALT (Structured Assertion Language for Temporal Logic) [12]. This analysis is fully automatic and the model checker will return a counterexample for any unsatisfied requirement, which helps to diagnose the underlying problem.

We augment static analysis with dynamic methods for early validation. Our tool contains a simulation engine [13] which enables step-wise execution of the system according to its formal semantics using (i) manually set input data, (ii) values obtained in random testing, (iii) data fed back by an environment model, or (iv) simulation based on traces of previous runs *on the target platform*.

The formal semantics not only facilitate bug hunting at model level, but also allow for early performance analysis, automatic computation of allocations of distributable software components to the hardware platform given in the TA, and automatic scheduling. Furthermore, executable code and platform configuration are generated automatically. These techniques are detailed in the following.

System Partitioning and Code Generation. Using the information of both LA and TA, automatic system synthesis can be performed. Initially, code for each distributable software component specified in the Cluster Architecture is

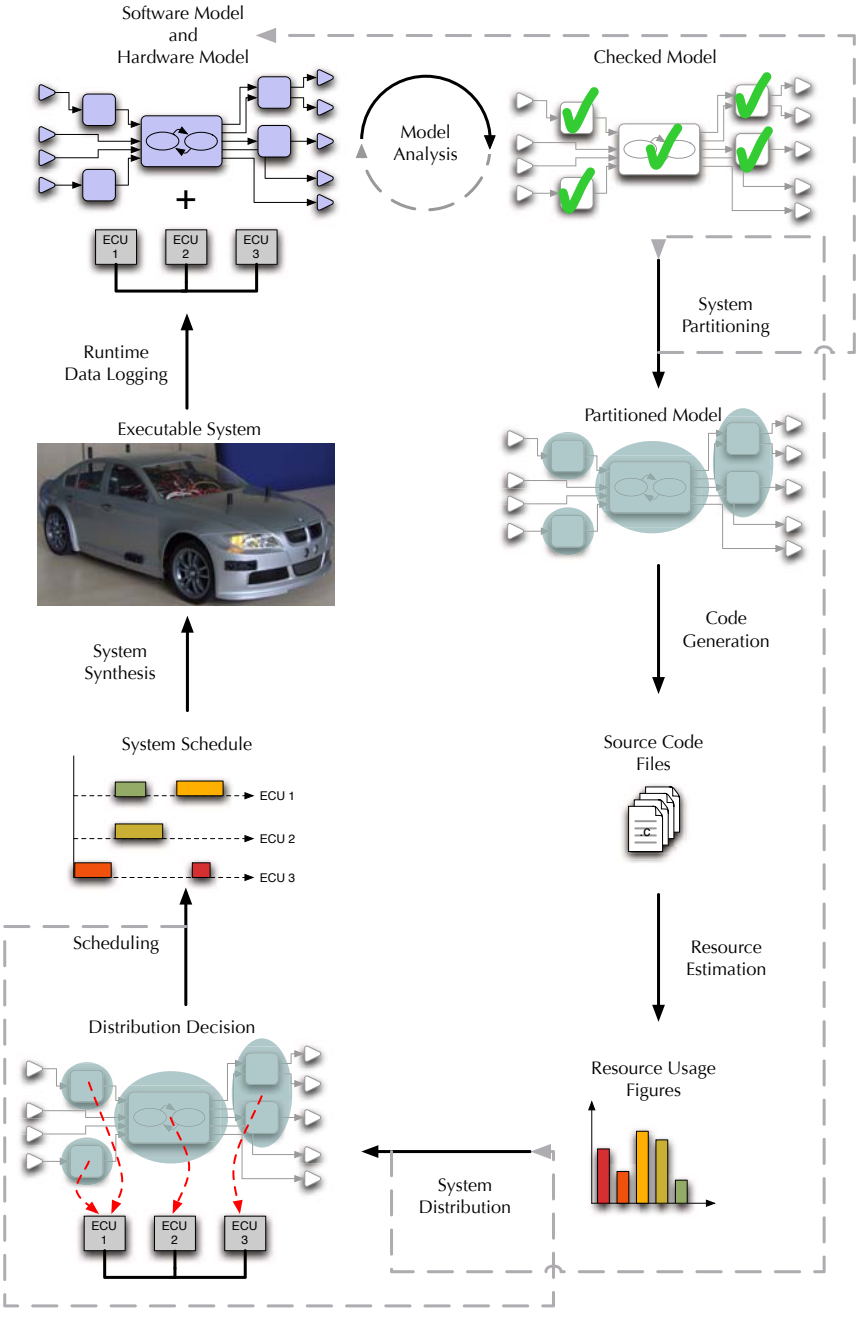


Fig. 2. Artifacts generated along the development process; solid arrows depict regular workflow while dashed arrows indicate optional refinement iterations

generated [14]. To test correctness of code generation, we also implemented an I/O conformance tester building upon the TorX ioco test framework [15].

Calls to a middleware realize both interaction of distributed software components and interfacing with sensors and actuators. These calls are injected into the code during system synthesis, as introduced in [16]. The middleware also retains the state of the components between cyclic invocations and provides a global clock for timely execution of time-triggered tasks.

Resource Estimation. The resource requirements of each distributable software component have to be evaluated for subsequent distribution and scheduling decisions. Automated performance estimation calculates the worst-case execution time and memory consumption of each component. Using the *SciSim* [17] framework, the generated code is instrumented and analyzed to compute its resource usage for every processing node the component might be executed on.

System Distribution. Based on this resource estimation, an optimal placement of distributable software components to run on computing nodes modeled in the Hardware Architecture is computed (cf. [18]). We use an integer linear programming solver to determine a solution yielding minimal cost under the hardware capability constraints. This includes their processing power and memory, as well as the communication systems interconnecting the nodes. It may turn out that no distribution of components exists that satisfies all constraints. In that case another refinement iteration is necessary as depicted in Figure 2.

Scheduling. Once a placement of software components has been determined, a suitable schedule is computed for each processing node. In doing so, data- and control flow dependencies must be considered, using the technique described in [19] to guarantee preservation of semantics on the target system. We use a satisfiability modulo theories (SMT) solver to determine a valid schedule. The result guarantees that starting times are always greater than finishing times plus communication delays of all components depended on. Again, the system distribution refinement process is initiated, if there is no feasible solution.

For implementing the computed schedules, and to preserve the synchronous semantics of COLA, we rely on time-triggered, non-preemptive execution, which must be supported by the employed operating system. In addition, the use of a global time-triggered schedule facilitates the definition of system-wide operating modes which are switched synchronously, as presented in [20].

Platform Configuration. Being a synchronous language, COLA assumes the model to be cyclically executed at discrete points in time. To preserve the models' semantics down to a concrete implementation, this assumption is safely approximated using a time-triggered schedule for execution of software components. To guarantee execution times calculated in the scheduling step, each component must be processed without being interrupted. Thus the employed operating system has to offer a non-preemptive scheduling algorithm.

We realize communication between software components, which are either co-located or allocated to different system nodes, in a generic middleware, as

introduced in [16]. Besides transparent communication, the middleware is responsible for storing each software component's state between cyclic invocations. Moreover, a global clock is provided by the middleware, enabling synchronized execution of software components according to the calculated schedule. Finally, the middleware features transparent hardware interaction for application tasks. To fulfill these duties, the middleware has to be configured according to the actual allocation of software tasks onto hardware nodes. This configuration file is generated as part of the code generation step. This defines the location of sensors and actuators as well as a mapping of exchanged messages to logical addresses.

Model Level Debugging. The employed middleware includes an optional logging mechanism, which allows logging of runtime data in the target system. For each cluster the logging facility may be activated in the design model. The middleware then retains all input and output data, as well as the internal state of the cluster, and stores them for later review. These data can subsequently be downloaded to a development computer and imported by our simulator for offline inspection. This technique enables model level debugging of systems designed with COLA, thus making classical debugging at source code level, e. g., using screen outputs or remote debuggers, redundant. Using this option, the presented approach closes the development circle depicted in Figure 2, allowing round-trip engineering for successive design improvements.

3.3 Tool Integration

In order to enable seamless integration of the tools, we maintain all models in a central *repository*. The models required for development are based upon an explicit metamodel that defines the syntax of the integrated modeling language. This metamodel enables uniform, homogeneous access to all models, and therefore eases the definition of model transformation steps. Furthermore, the central model repository prevents redundancy, and allows to check consistency between models. This comes at the price that tools have to be re-implemented from scratch to follow the desired paradigm. Thus, we also envision a framework that eases the development of tools and provides cross-cutting functionality like configuration management. We will report on this framework in future work.

We implemented a *front-end* that allows the engineer to access the models in the repository using the concrete syntax provided by the integrated modeling language. To this end, the front-end provides editors that present the models in graphical, textual, or tabular notation. The implementation builds upon the Eclipse platform, the plug-in architecture of which permits the extensibility of the tool. We have implemented a number of plug-ins that realize different steps in our seamless modeling process as described in the preceding section. Figure 3 gives an impression of the front-end we developed for COLA.

In addition, we realized a *back-end* to handle resource intensive steps like verification. Time-consuming tasks are moved to the back-end in order to not affect the performance and responsiveness of the engineer's workstation (front-end). The back-end should also be used to perform continuous analysis and

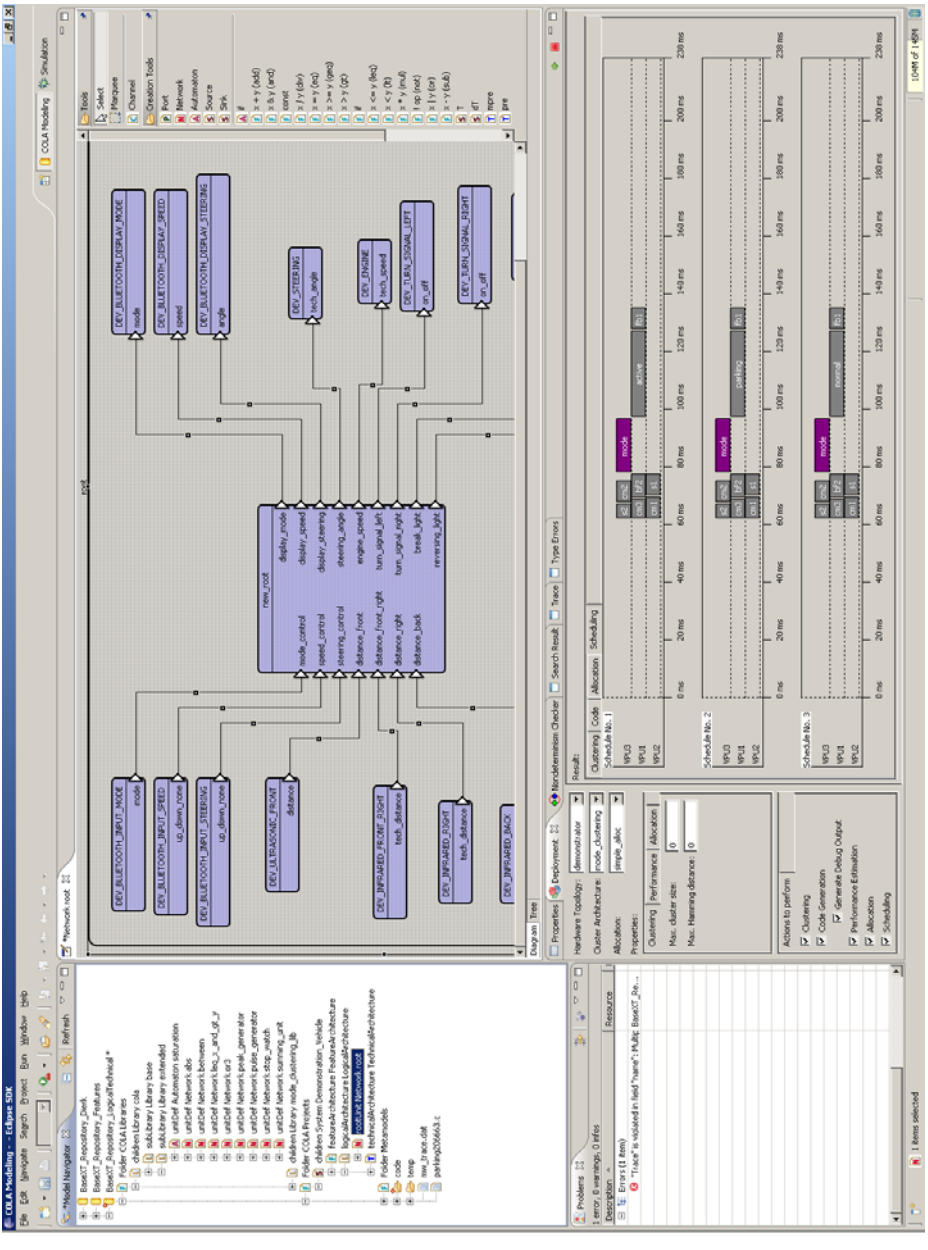


Fig. 3. Front-end of the integrated COLA engineering tool

integration of the models. This approach enables the early detection of errors, and helps to guarantee the quality of the resulting system.

The seamless development process is operationalized by a separate *process engine*. The process engine controls the creation of the models in that it coordinates

the process activities carried out by different stakeholders. In combination with the model repository, it enables the systematic, distributed development of a system. In order to tailor it to specific requirements, the process engine is parametrized by an explicit process model. The process model defines the activities and how they have to be orchestrated. For each process step a distinct set of model consistency constraints is defined as part of the process model. These ensure proper transitions to subsequent activities.

4 Case Study

We realized a case study, using the concepts and tools described before, to prove the viability of our approach. We built the model car shown in Figure 4 and implemented an autonomous parking system based on several distance sensors on this platform. Additionally, the system is controllable manually via Bluetooth connection to a cell phone. Furthermore, it should initiate an emergency stop when reaching a given minimum distance to obstacles.

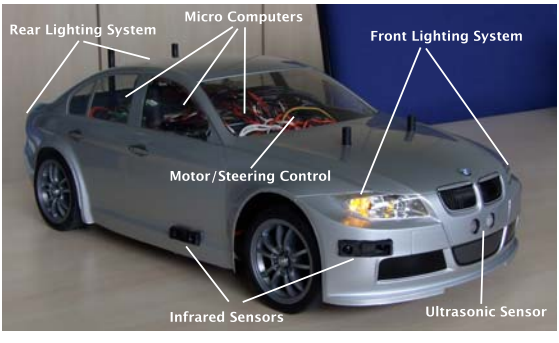


Fig. 4. Model car used in the case study on a scale 1:10

The model car is equipped with three Gumstix® micro computers connected through an Ethernet network. The previously mentioned middleware is employed for data exchange and clock synchronization services. Distances are measured using two infrared and one ultrasonic sensor. Bluetooth is used as another input, connected to the cellular phone. As actuators we have the model car's motor and steering, as well as indicator, reversing, and breaking lights.

We specified the desired system using the proposed architectural levels from FA over LA to TA and were able to generate code for the specified software components as well as the middleware configuration files. The according COLA model consisted of 37 automata and 225 networks. These elements were then partitioned into 11 distributable clusters. Our code generator transformed the model into 14 C code files with a total of 6089 LOC, and the configuration file for our middleware. After cross-compilation of the code, the system behaved correctly without the need of any manual changes.

5 Related Work

During the last years both academic and industrial research and development projects were engaged in model-based development for safety-critical embedded systems. Yet, there is no fully integrated tool which covers the complete development process. Such a process should start with requirements specification in a formal way, proceed with the behavioral system design phase, and finally result in an automatically deployed and configured product. After relating the concept presented in this paper to commercial off-the-shelf products, we proceed to comment on projects from academia.

Of course, there are tools for each of the mentioned phases. IBM's Rational DOORS, and Reqtify from Geensys are widely used requirement specification and management tools. MATLAB/Simulink/Statechart from The MathWorks, SCADE by Esterel Technologies which is based on the synchronous data-flow language Lustre, and ASCET-SD from ETAS cover system and software design. Finally, for target code generation, e. g., the Real-Time Workshop Embedded Coder from The MathWorks or the KCG code generator by Esterel Technologis are employed. The mentioned tools have varying levels of integration: those of the same vendor are highly integrated, those from different vendors have an integration which is often restricted to synchronization mechanisms or rely on third party adapters like OmniLink by Omniteam. For the latter, neither an integrated data model is used nor a 100% data compatibility is guaranteed.

The EU project DECOS [21] is similar to the presented approach in a way that they aim also at the development of dependable embedded systems. They present a tool-chain of more or less integrated existing tools like SCADE for behavioral modeling as well as TTPlan and TTPbuild by the TTTech Computertechnik AG for configuration and scheduling. Neither a central repository to store *all* artifacts created during the development process nor a formalization of requirements in the early development phase are considered.

Similar to the our work, do Nascimento et al. propose to separate specification of platform independent (PIM) (comparable to our LA) and platform specific (PSM) (comparable to our TA) models, however using different metamodels [22]. Currently, they solely support UML for behavioral modeling which—according to Broy [2]—does not cater for the specific needs of the domains of embedded systems design, e.g., in the automotive industry. The *Systems Modeling Language* (SysML) [23] tries to overcome this limitation, by restricting UML to only essential language constructs. In return, it adds for instance requirement diagrams to efficiently capture, e. g., functional requirements. The UML2 Profile for *Modeling and Analysis of Real-Time and Embedded systems* (MARTE) [24], which is currently being standardized, enriches UML by concepts to support the development of real-time embedded systems. It captures aspects like hardware and software modeling as well as schedulability and performance analysis.

In [25], the authors present a layered approach to implement MATLAB/Simulink models via a translation to SCADE/Lustre on the distributed time-triggered platform TTA [26]. This work completely ignores formalization and management

of requirements, thus lacking support during early design phases and the possibility of automated reasoning about the designed models.

6 Conclusion

We have outlined and implemented an integrated modeling process that makes specification as well as implementation of embedded systems a controllable business. The procedural manner suggested here convinces with its agility as well as efficiency, that we believe to be beyond what can be achieved by applying a high level of automation only. The prerequisite for doing so is a well-defined formal integrated modeling language that covers all process steps. It empowers the developer to implicitly derive a mathematical representation of the modeled artifacts as a designated side effect. Several design problems can then be solved automatically by utilizing computational support. We were already able to prove the stated benefits, using our integrated modeling tool during the realization of the described case study.

References

1. Schulz, S., Rozenblit, J.W., Buchenrieder, K.: Multilevel testing for design verification of embedded systems. IEEE Design & Test of Computers 19(2), 60–69 (2002)
2. Broy, M.: Automotive software and systems engineering (panel). In: MEMOCODE, pp. 143–149 (2005)
3. Arthur, S., Breed, H.N., Schmitt-Luehmann, C.: Shifting car makeup shakes up OEM status quo: Software strength is critical. IBM White Paper (2003), http://www.ibm.com/services/in/igs/pdf/g510-1692-00-shifting-car-makeup-shakes-up-oem-status-quo.pdf (last access: 2009-07-13)
4. Broy, M., Feilkas, M., Herrmannsdoerfer, M., Merenda, S., Ratiu, D.: Seamless model-based development: From isolated tools to integrated model engineering environments. Proceedings of the IEEE 98(4), 526–545 (2010)
5. Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E., Peterson, A.S.: Feature-oriented domain analysis (FODA) feasibility study. Technical report, Software Engineering Institute, Carnegie Mellon University (1990)
6. Halbwachs, N., Caspi, P., Raymond, P., Pilaud, D.: The synchronous data-flow programming language LUSTRE. Proceedings of the IEEE 79(9), 1305–1320 (1991)
7. Pretschner, A., Broy, M., Krüger, I.H., Stauner, T.: Software engineering for automotive systems: A roadmap. In: FOSE, pp. 55–71 (2007)
8. Broy, M., Feilkas, M., Grünbauer, J., Gruler, A., Harhurin, A., Hartmann, J., Penzenstadler, B., Schätz, B., Wild, D.: Umfassendes Architekturmodell für das Engineering eingebetteter Software-intensiver Systeme. Technical Report TUM-I0816, Technische Universität München (2008)
9. Kugele, S., Tautschnig, M., Bauer, A., Schallhart, C., Merenda, S., Haberl, W., Kühnel, C., Müller, F., Wang, Z., Wild, D., Rittmann, S., Wechs, M.: COLA – The component language. Technical Report TUM-I0714, Technische Universität München (2007)

10. Rittmann, S.: A methodology for modeling usage behavior of multi-functional systems. PhD thesis, Technische Universität München (2008)
11. Kühnel, C., Bauer, A., Tautschnig, M.: Compatibility and reuse in component-based systems via type and unit inference. In: SEAA, pp. 101–108 (2007)
12. Bauer, A., Leucker, M., Streit, J.: SALT—structured assertion language for temporal logic. In: Liu, Z., He, J. (eds.) ICFEM 2006. Bauer, A., Leucker, M., Streit, J, vol. 4260, pp. 757–775. Springer, Heidelberg (2006)
13. Herrmannsdoerfer, M., Haberl, W., Baumgarten, U.: Model-level simulation for COLA. In: MISE, pp. 38–43 (2009)
14. Haberl, W., Tautschnig, M., Baumgarten, U.: From COLA Models to Distributed Embedded Systems Code. IAENG International Journal of Computer Science 35(3), 427–437 (2008)
15. Tretmans, J., Brinksma, E.: TorX: Automated model-based testing. In: First European Conference on Model-Driven Software Engineering (2003)
16. Haberl, W., Birke, J., Baumgarten, U.: A Middleware for Model-Based Embedded Systems. In: Halperin, D., Mehlhorn, K. (eds.) ESA 2008. LNCS, vol. 5193, pp. 253–259. Springer, Heidelberg (2008)
17. Wang, Z., Sanchez, A., Herkersdorf, A.: Scisim: a software performance estimation framework using source code instrumentation. In: WOSP, pp. 33–42 (2008)
18. Kugele, S., Haberl, W., Tautschnig, M., Wechs, M.: Optimizing automatic deployment using non-functional requirement annotations. In: ISoLA, pp. 400–414 (2008)
19. Kugele, S., Haberl, W.: Mapping Data-Flow Dependencies onto Distributed Embedded Systems. In: SERP, pp. 272–278 (2008)
20. Haberl, W., Kugele, S., Baumgarten, U.: Reliable operating modes for distributed embedded systems. In: MOMPES, pp. 11–21 (2009)
21. Herzner, W., Schlick, R., Schlager, M., Leiner, B., Huber, B., Balogh, A., Csertan, G., LeGuennec, A., LeSergent, T., Suri, N., Islam, S.: Model-based development of distributed embedded real-time systems with the decos tool-chain. In: SAE Aerotech (2007)
22. do Nascimento, F.A.M., Oliveira, M.F.S., Wagner, F.R.: Modes: Embedded systems design methodology and tools based on mde. In: MOMPES, pp. 67–76 (2007)
23. Object Management Group: Systems Modeling Language (SysML). OMG document: v1.1-08-11-01.pdf (2008)
24. Object Management Group: UML profile for modeling and analysis of real-time and embedded systems (marte), beta 2. OMG document: ptc/08-06-07 (2008)
25. Caspi, P., Curic, A., Maignan, A., Sofronis, C., Tripakis, S., Niebert, P.: From simulink to SCADE/lustre to TTA: a layered approach for distributed embedded applications. In: LCTES, pp. 153–162 (2003)
26. Kopetz, H.: Real-Time Systems: Design Principles for Distributed Embedded Applications. Kluwer, Dordrecht (1997)

Timely Time Estimates

Andreas Holzer[1], Visar Januzaj[2], Stefan Kugele[3], and Michael Tautschnig[1]

[1] Technische Universität Wien
AB Formal Methods in Systems Engineering
Favoritenstr. 9, 1040 Wien, Austria
{holzer,tautschnig}@forsyte.at
[2] Technische Universität Darmstadt, Fachbereich Informatik,
FG Formal Methods in Systems Engineering - FORSYTE,
Hochschulstr. 10, 64289 Darmstadt, Germany
januzaj@forsyte.de
[3] Technische Universität München, Institut für Informatik,
Boltzmannstr. 3, 85748 Garching bei München, Germany
kugele@in.tum.de

Abstract. Estimations of execution time are essential for design and development of safety critical embedded real-time systems, such as avionics, automotive and aerospace systems. In such systems, execution time is part of the functional specification, hence correct behaviour requires sufficiently powerful target hardware to meet deadlines or achieve required polling rates, etc. Yet, grossly overestimated resource usage results in excessive cost per unit. For a proper choice of the target platform, qualitatively good execution time estimates are required at an early stage of the development process.

In this paper we propose a framework which provides software engineers with execution time estimates of the software under development in a demand-driven manner, i. e., the engineers ask for timing information at program or function level with respect to different target hardware platforms. In a platform-independent manner we extract the necessary information from the code and combine it with platform-specific information, resulting in the time estimate. We implemented our framework on top of the test input generator FShell and its query language FQL. Preliminary experiments on C code show the viability of our approach.

1 Introduction

During the last decades, embedded systems have become ubiquitous in everyday life. They are used in consumer electronics devices as well as in highly safety-critical areas such as medical equipment, cars, and aeroplanes. Especially in such safety-critical systems, *time* plays an important and crucial role: correct behaviour requires timely execution. In the automotive domain, for example, an airbag system has to inflate in a given amount of time to perform its mission-critical task successfully. In the avionics domain, during flight the wings have to be stabilised periodically in a millisecond range, in order to avoid an undesired build up (resonance disaster), which endangers a stable and safe flight.

T. Margaria and B. Steffen (Eds.): ISoLA 2010, Part I, LNCS 6415, pp. 33–46, 2010.

Development of large-scale real-time systems, as found in the automotive and aerospace domains, is performed by a sizeable number of stakeholders. Consequently, each engineer has only limited, if any, access to the target platform at early stages of the development process. In these early phases, however, execution time information is essential—especially in the context of hardware/software co-design—to select suitable hardware, i. e., processors, memory, and other peripherals. As the system is still expected to evolve, however, execution time of software components can at best be an estimate. Still, it is crucial that these estimates assumed in early development phases are as precise as possible with regard to the complete executable system: According to [1,2], the cost for complete system redesign or larger system changes due to functional errors increases considerably at later stages of the development process. Gustafsson et al. [3] argue that it behaves similarly for errors in the time domain, if wrong assumptions were made right at the beginning.

In this paper we present an approach that enables the software developer to obtain estimates of the timing behaviour of the software under development without direct access to the target platform. Moreover, the engineer can ask for timing estimates not only for whole programs, but even at the granularity of single functions. This is advantageous if the entire system has not yet been completed, but time estimates of subsystems or functions are of interest. This paper is a follow-up to our work in progress paper [4], describing the realisation and experiments.

In our approach, we first run several commonly used benchmarks on potential target platforms. We thereby obtain the platform specific execution time information (hardware profile). This is the only point in time where we require access to the target hardware. Given sufficient information from hardware manufacturers, however, even the hardware profile could be synthesised. These profiles are stored in a repository for later reuse. Once the platform specific information has been assembled, time estimation is performed solely on the development workstation. The process of obtaining time estimates is easy for the developer and comprises the following three steps: (i) specification of the source code or function(s) to be analysed, (ii) stating quality criteria, i. e., on which basis (statement coverage, or path coverage, etc.) the time estimate is to be computed, and (iii) choice of the intended execution platform. In order to obtain time estimates for the code range specified in step (i), the source code is first translated into the LLVM [5] intermediate representation (bytecode program) and is then instrumented. For steps (i) and (ii) the software engineer uses the query language FQL [6]. FQL has been designed for software developers and ease of use, and comes with a back-end such as FShell [7,8], which *automatically* generates test inputs and a test harness for the instrumented source code. For each generated test input, the test harness including the instrumented program is executed. During program execution instruction counts are obtained by counting the number of each instruction type of the LLVM intermediate representation. For each instruction type, e. g., `ADD`, `MUL`, or `RET`, weights have to be determined. This is done by selecting appropriate benchmarks from the repository and computing their weights. Finally, the sum of the weighted instruction counts yields the time estimate.

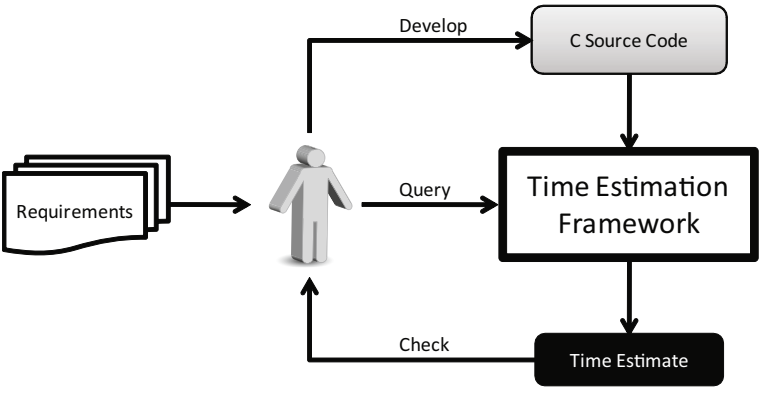


Fig. 1. Software engineer using the framework

Organisation. In Section 2 we describe our framework in detail. We discuss preliminary experiments in Section 3. We relate our approach to existing work in Section 4 and conclude in Section 5.

2 Framework

The idea behind our approach is illustrated in Figure 1. Integrating our framework in the software development process allows the software engineer to observe the timing behaviour of the software being *developed* with respect to the given (time) requirements. This means, the software engineer can *query* (ask questions) about the execution time of the software when run on a particular target platform. In response to the query the framework yields a time estimate, which can then be *checked* against the corresponding requirements. If the requirements are not satisfied, the software can be improved or an alternative target platform may be chosen, until the requirements are finally met. The software engineer can state queries about the whole software or parts of it, such as functions, particular paths in the code/function or even simple code chunks. This flexibility allows the engineer to check, for instance, time estimates relative to different system modes, as can be found in automotive/avionics systems, e. g., *accelerating*, *taking off*, *landing* or *braking*.

Figure 2 depicts the high-level structure of our framework. Upon input of *C source code*, the *code unit* to be analysed, e. g., a function in the given source code, and the choice of a *target platform*, we automatically compute an estimate of the execution time. To do so, we first translate the choice of the code unit into an FQL statement. Together with a coverage specification chosen by the engineer we compute *test inputs* and automatically build an adequate *test harness*. Running the original program using this test harness will yield executions that (a) achieve the specified code coverage and (b) are guaranteed to pass through the specified code unit. For successive time estimation, however, we actually run an *instrumented* version of the program that contains *instruction counters*. The

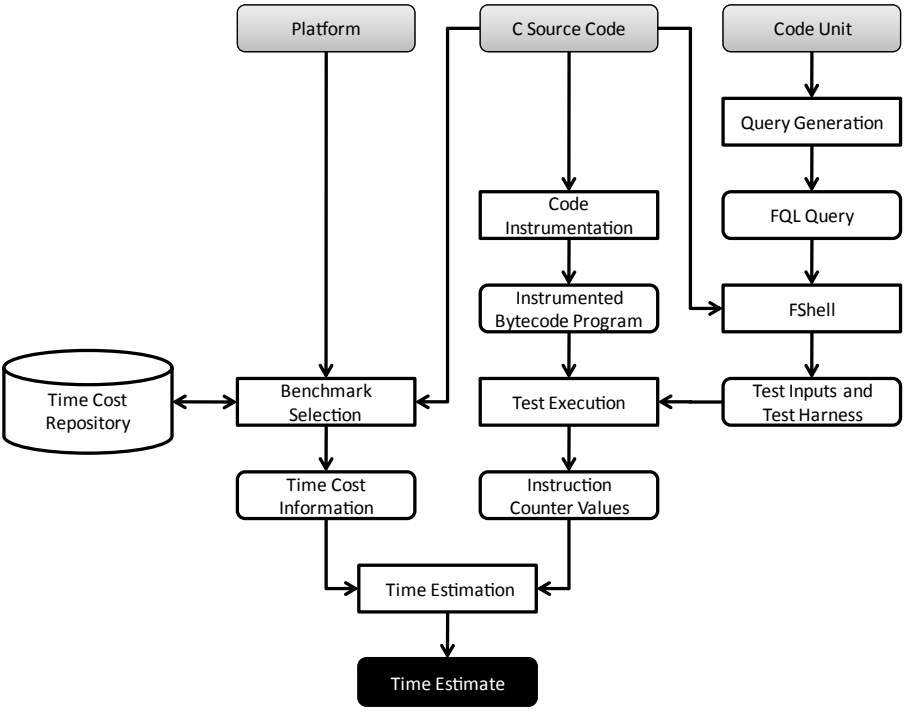


Fig. 2. Time Estimation Framework

time estimator combines the valuations of these counters collected after each execution with platform-specific instruction timing information to compute the desired *time estimate*. In the following we describe each of these aspects in detail.

2.1 Code Instrumentation

We use the LLVM compiler framework [5] for *code instrumentation* to perform a translation of the source code into the LLVM bytecode intermediate representation. LLVM offers a modified version of the GNU C compiler that allows the inclusion of platform-independent optimisations during the translation step. This narrows the gap between high-level C code and the instructions that are actually executed on the target platform, which supposedly improves later time estimates. On the LLVM bytecode level we introduce *instruction counters*, i. e., variables that keep track of the number of instruction executions for each instruction type, e. g., `ADD`, `MUL`, `JMP`, or `CALL`. Upon execution the instrumented bytecode yields the *occurrence set*, consisting of tuples (c_n, i_n), where c_n is the instruction counter value corresponding to the n-th LLVM instruction i_n, e. g., (9, `ADD`) meaning that the instruction `ADD` is executed 9 times. In our implementation, these counters can be recorded for each function invocation, i. e.,

the number of the executed instructions for each function run can be tracked separately.

2.2 Test Input Generation and Test Harness

Our test input generator, FShell [7,8], uses FQL [6], which we briefly describe below, as query language: The user describes test suites using FQL, and FShell efficiently computes adequate test inputs. These test inputs yield, upon execution of the program on such a set of inputs, the desired code coverage. For automated execution of the program on the computed test inputs FShell includes an automatic test harness builder. Such a test harness is effectively a wrapper, given as C code, that executes the function under test with the computed inputs.

FQL queries describe test suites over source code in a declarative way and have the general form `cover` A `passing` B. Therein, A is a so-called *coverage pattern* and B is a *path pattern*. Both types of patterns build on the concept of *filter functions*, which provide a means to identify parts of the source code, e. g., the filter function `@FUNC(foo)` refers to the source code in function `foo`, whereas `ID` refers to the source code as a whole. There are also filter functions referring to code structures needed by standard coverage criteria like basic block coverage or condition coverage: `@BASICBLOCKENTRY` yields all basic block entries and `@CONDITIONEDGE` yields all evaluations of conditions. We can also refer to the entry and exit of a C function `f` using the filter functions `@ENTRY(f)` and `@EXIT(f)`, respectively.

FQL enables the user to refer to program locations, program edges, or bounded program paths inside of program parts identified by filter functions by the keywords `NODES`, `EDGES`, and `PATHS`, respectively. A program edge is a transition between two program locations annotated with a statement. For example, the expression `EDGES(@LABEL(L))` refers to the edges in the code annotated with code label `L`. The expression `PATHS(ID, 2)` denotes all program paths, i. e., sequences of program edges, where no edge occurs more than twice.

Using expressions of the form `NODES(T)`, `EDGES(T)`, and `PATHS(T, k)` as alphabet, we formulate regular languages whose words describe program executions, e. g., the program executions denoted by `EDGES(ID)*.EDGES(@LABEL(L)).EDGES(ID)*` pass after finitely many steps an edge annotated with code label `L`. Here, '.' denotes concatenation and '*' the Kleene star. Additionally, FQL contains the operator '+' representing an alternative. Since `EDGES` is used commonly, FQL allows to omit it, e. g., the pattern above can be given as `ID*.@LABEL(L).ID*`.

Coverage and path patterns are regular languages as given in the preceding paragraph. Every word in a coverage pattern denotes a test goal and since a test suite has to be finite, coverage patterns do not permit the application of the Kleene star operator. But, one can introduce path patterns as new alphabet symbols in coverage patterns using *quoting*: The query

```
cover "ID*".@LABEL(L)."ID*"
```

requests for a test suite that covers all program edges annotated with code label `L` preceded and succeeded by an arbitrary number of program edges. Here,

by quoting, the pattern `"ID*"` does not introduce an infinite number of words but yields only one new alphabet symbol.

The semantics of a query `cover` A `passing` B requires a test suite which (i) contains for each word in A, called *test goals*, at least one matching test case, and (ii), contains only test cases which are matched by B. For example, to achieve decision coverage with the constraint that line 17 is reached at least once we use

```
cover "ID*".@DECISIONEDGE."ID*" passing ID*.@LINE(17).ID*
```

The integration of FQL/FShell into our time estimation framework allows the user to write queries at the level of small code chunks, such as functions or even paths within functions, yielding corresponding test inputs which are used for time estimation. In this way the user can inspect the timing behaviour with respect to system modes, e. g., *take off* or *landing* in case of avionics software. The developer may then fine-tune their software to meet the corresponding requirements.

```
1  int min(int a, int b, int c)
2  {
3    int m;
4    if (a <= b) {
5      if (a <= c) m = a;
6      else m = c;
7    } else {
8      if (b <= c) m = b;
9      else m = c;
10   }
11   return m;
12 }
```

Listing 1. Sample program

As an example consider the code in Listing 1, which returns the minimal value of three integer variables. In case we want to obtain a time estimate for function min, we first determine a test suite covering, e. g., all basic blocks of min. For this purpose, we state the query

```
cover "ID*".(@FUNC(min) & @BASICBLOCKENTRY)."ID*"
```

Given this query and the source code of Listing 1, FShell computes four different sets of test inputs, i. e., evaluations of parameters a, b, and c, in order to cover all basic blocks in min (cf. Table 1).

Table 1. Example test suite

Test Case	a	b	c
1	44437522	35655690	44437522
2	0	2	−2147483645
3	1207959552	1140850690	1073741828
4	0	1073741826	4

```
1 int main(int argc, char* argv[])
2 {
3   long tc = -1;
4   if(argc != 2)
5   {
6     printf("Expected test case id as single argument\n");
7     return 1;
8   }
9   errno = 0;
10  tc = strtol(argv[1], NULL, 10);
11  if(errno != 0)
12  {
13    printf("Failed to parse test case id\n");
14    return 2;
15  }

17  switch(tc)
18  {
19    case 1:
20      min(44437522,35655690,44437522);
21      break;
22    case 2:
23      min(0,2,-2147483645);
24      break;
25    case 3:
26      min(1207959552,1140850690,1073741828);
27      break;
28    case 4:
29      min(0,1073741826,4);
30      break;
31    default:
32      printf("No test case available for id %ld\n", tc);
33      return 3;
34  }

36  return 0;
37 }
```

Listing 2. Test harness

Subsequently a test harness is derived. The result for these four test cases is shown in Listing 2. The test harness is a main function that serves as a wrapper which calls function min according to the selected test case. In our approach the generated test harness is then linked to the instrumented version of function min (cf. Section 2.1). By running the resulting executable with the different test inputs, we obtain values for the instruction counters (cf. Table 2) and are able to determine time estimates for every test case (cf. Section 2.5).

Table 2. Instruction counters resulting from generated test suite

LLVM Instruction	TC 1	TC 2	TC 3	TC 4
Load	19	31	19	31
Br	9	20	9	20
Alloca	24	36	24	36
Store	19	26	19	26
ZExt	4	11	4	11
BitCast	4	6	4	6
ICmp	11	22	11	22
Select	3	0	3	0
Ret	1	1	1	1

2.3 Time Cost Repository and Hardware Benchmarking

The time cost repository can be filled in different ways: one way is to extract the time cost information from data sheets [9]. This approach is practical for timing (cycle) information of existing processors but remains problematic with respect to the time estimation accuracy, since pipelining and cache effects are not considered. Furthermore, the mapping between LLVM instructions and actual processor instructions requires a deeper hardware knowledge. Another way (which we apply on our approach) is to collect the time cost information by running benchmark programs on the target platform, cf. [9,10]. The time cost repository stores the results of selected benchmarks, e. g., from Mälardalen [11] and SPEC [12] benchmarks for each target platform.

More precisely, the repository holds information about the benchmarking results of each available target platform, i. e., for each target platform the time cost of each executed benchmark is stored, e. g., for benchmarks B_1 and B_2 with execution times $t_{B_1} = 15ms$ and $t_{B_2} = 7ms$, respectively, measured on platform P there is an entry

$$P : \{(t_{B_1} = 15ms), (t_{B_2} = 7ms)\}$$

in the repository. Furthermore, for each benchmark B_i the number and the type of the executed LLVM instructions, the occurrence set, is stored, e. g.,

$$B_i : \{(13, \texttt{ADD}), (15, \texttt{MUL}), \ldots, (1, \texttt{RET})\}$$

2.4 Benchmark Selection

Initially, we measure for each benchmark program B the execution time t_B for a specific test input and a specific target platform. Furthermore, we determine for B the number of executed LLVM instructions as described above. Thereby, we get for every LLVM instruction i the number of executions c_i. Since LLVM instructions are not executed directly, but are translated into machine code during the compilation phase this approach introduces a source of inaccuracy into our framework.

We start with the hypothesis that the equation

$$c_1 \cdot w_1 + \ldots + c_n \cdot w_n = t_B$$

holds for *weights* w_i. Having fixed the instruction counts c_i for $1 \leq i \leq n$, and the execution time t_B by measurements on the actual target platform, it remains to determine the instruction weights w_i. To constrain the solution space for these weights we consider several benchmarks at once. Furthermore, we want to ensure that each weight is strictly positive. In general, however, this system of linear constraints will be infeasible: The model does not capture effects of low-level compiler optimisations, cache effects, or sublinear execution times through pipelining. Therefore, we accept a certain error in our equations and use the method of steepest descent (with respect to the deviation of measured and calculated execution times) to determine the weights (starting at a point where all weights are initially equal) while ensuring the mentioned additional constraints. Later the resulting weights are used for time estimation and thus form our time cost repository. For each target platform we have an additional entry in the repository with the set of instruction weights, e. g., for the instruction weights $w_1 = 0.001$ for `ADD` and $w_2 = 0.002$ for `MUL` on platform P we have

$$P : \{(0.001, \texttt{ADD}), (0.002, \texttt{MUL})\}$$

In our current setting, we cluster benchmarks, which yields weights with small error margins (at maximum 15 percent deviation from the actual execution time), and select for a given source code the weights according to the matching cluster.

2.5 Time Estimation

Once the occurrence sets corresponding to the user's query are determined, these counter values are multiplied with the computed weight and summed up, yielding the estimated execution time of the code unit being analysed. The weights are the time cost information taken from the time cost repository, respectively calculated regarding the benchmark selection, and are specific for each instruction type.

Figure 3 illustrates the time estimation process. After FSHELL has calculated the test inputs and the corresponding test harness, we integrate each test input separately into the instrumented bytecode program and execute it. Each execution yields an occurrence set O_j, $1 \leq j \leq m$, where m is the number of all test inputs, thus for each execution we get the instruction counter values of the executed instructions with respect to the test input j. We multiply each counter value c_i of the occurrence set O_j with the corresponding weight w_i from the time cost information and the overall sum represents the estimated execution time of the code unit regarding the current test input j, thus we have:

$$time(O_j) = \Sigma_{i=1}^{n} c_i \cdot w_i.$$

We calculate similarly for each execution the corresponding time estimate and select the maximal estimate, which represents the final time estimate of the code unit under analysis.

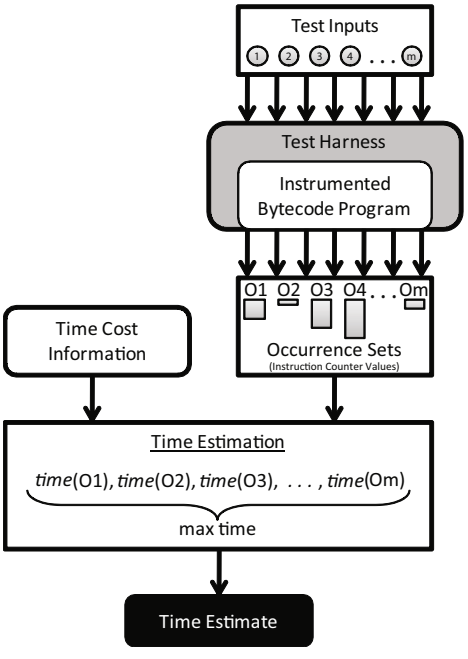


Fig. 3. Time estimation

3 Experiments

In our experiments, we obtained the hardware profile for an AVR[1] and a MIPS[2] platform by executing programs from the WCET benchmark suite [11], on these systems. After measuring their execution times we instrumented these programs, as described in Section 2, and determined the occurrence sets for the benchmarks. Before determining the weights for the instructions (see Section 2.4), we clustered the benchmark programs with respect to common characteristics of the source code [9] in order to get meaningful values within each cluster. Then, for every benchmark cluster, we derived weights. These coefficients in combination with the clusters constitute the time cost repository for our experiments.

To check the quality of the calculated coefficients, we used further programs and determined a time estimate for them using coefficients from their corresponding clusters. The errors of the time estimates, compared to the execution time measured on the real platforms, were below 15 percent (cf. Table 3). The time estimates showed over- as well as underapproximation. This behaviour occurred even for the same benchmark when estimating time for different target platforms.

[1] AVR32B rev. 1, AT32AP700x rev. A, 140 MHz, i-cache: 16K, d-cache: 16K.
[2] CUST_WSX16 (CN3860p3.X-500-EXP), Cavium Octeon V0.3.

Table 3. Time estimates for benchmarks

Benchmark	Estimated Execution Time [ms]		Actual Execution Time [ms]		Error [%]	
	AVR	MIPS	AVR	MIPS	AVR	MIPS
DUFF	39.6	8.9	45.0	8.0	12.0	11.3
EDN	2433.9	526.1	2550.0	460.0	4.6	14.4

4 Related Work

We compare our work to existing approaches for early resource estimation with a focus on the two aspects of hardware modelling and execution time computation. For an overview of a wide range of techniques for worst-case execution time (WCET) analysis cf. [13].

Most approaches use variations of Implicit Path Enumeration (IPET) [14] to compute execution times of a given piece of software. In IPET, a program is described as a system of linear inequalities that describe the control flow over the frequencies of basic blocks as variables. To compute the WCET an objective function is added that sums up the cost of each basic block multiplied by the frequency. Solving this linear programming problem for the maximum (minimum) yields the worst-case (best-case) execution time. The cost of each basic block can be determined in several ways. In its most basic variation the execution time of a basic block is assumed to be the sum of execution times of all contained statements [15]. Extensions of this schema to account for features of modern architectures have been proposed in [16]. In these approaches, the execution time of each statement is taken from hardware data sheets.

In [17] an alternative approach using measurements is proposed. The execution time of each (assembly-level) basic block is measured over a series of inputs to the program. Using the execution counts of each basic block as coefficients an equation system is formed and solved. Effects of (simple) pipelines are accounted for by adding extra variables for inter-basic-block effects. In contrast to this approach we determine execution times for each single statement to reuse this information for execution time estimation for yet unknown software in design space exploration.

In the area of hardware/software co-design effective design space exploration is well established and often based on simulation. For a discussion of problems of performing early performance validation for software-centric systems see [18]. In [19] a simulation-based approach was proposed for early hardware evaluation for embedded software-centric systems. They use simulation to estimate execution times on various target hardware architectures.

Early architecture evaluation is generally expected to yield results quickly. Hence, both simulation and costly WCET computation techniques are deemed inappropriate. Therefore Timing Explorer [20], which is based on AbsInt's aiT WCET analyser, only uses cheap (and less precise) analysis techniques. aiT requires detailed micro-architectural models. For early architecture evaluation Timing Explorer allows the user to easily configure models for new processors.

For the analysis it further requires a working compiler for the target architecture and reference code. Whereas aiT performs analysis at assembly code level, the approach proposed in [21] works on LLVM bytecode. Together with detailed processor models an analysis can be performed without the need for a working target compiler.

All the above approaches not based on measurements require (precise) micro-architectural models. Obtaining these is costly and time consuming, which hampers their applicability for early evaluation. Only recently an approach was presented which alleviates this problem the same way we do: Gustafsson et al. [3] propose the same technique for timing model identification using benchmarks and measurements. For estimation of execution times of single statements they suggest to build a linear equation system to be solved with regression methods. It will be implemented in the SWEET tool (SWEdish Execution time Tool). Whereas we use LLVM bytecode, they use ALF (ARTIST2 Language for WCET Flow Analysis) [22] as their analysis target language. Computation of WCET of software systems is subsequently performed using IPET, whereas we also use instrumentation and concrete execution to also compute execution times other than best/worst-case scenarios.

Such instrumentation-based approaches were also described in [9,23]: These approaches use counters for an abstract instruction set, which is then translated into machine-specific cycle-accurate information. To achieve high precision, this requires a detailed micro-architectural information, which we can avoid using our hardware benchmarking approach.

5 Conclusion

We have presented a framework for demand-driven time estimates. The combination of an automatic test input generation approach with a platform-independent timing estimation approach enables the continuous availability of timing information which can be gathered in a systematic and simple manner.

The time estimates generated by our framework cannot represent the exact execution time of the software being analysed, but provide good estimates that can help the developer to observe/monitor the behaviour of the software under development. The calculation of time estimates relies on the quality and type of the benchmarks used for platform benchmarking. In order to get the best benchmark representatives, we perform a software characteristics check, i. e., we choose only the benchmarking data of those benchmarks that have a similarity to the software under investigation. For instance, the benchmarking data of benchmarks performing mostly arithmetic operations is hardly suitable for software that performs extensive memory operations. This characteristics check increases the accuracy of the time estimates.

We have to stress here that the goal of our approach is not to replace a detailed timing analysis, which is normally performed after the software development process is completed. Instead we aim at providing the software engineer with a framework which is meant to be used at an early development stage, where exact

timing analysis are impossible, time consuming and too expensive. This shall aid to reduce time to market and possible rework cost. Our early time estimates are indeed not as exact as the actual execution time of the software on the target platform. They can, however, serve as a guide for the software engineer to make reasonable design decisions and to get an idea of how the software will behave on a specified target platform. This ensures that development is moving towards a suitable end product.

Our preliminary experiments show the principal viability of our approach, yet, a more rigid investigation in the context of a complex software development process has to be carried out to assess its impact on industrial development practice.

References

1. Boehm, B.W.: Software Engineering Economics. Prentice-Hall, Englewood Cliffs (1981)
2. Westland, J.C.: The cost of errors in software development: evidence from industry. Journal of Systems and Software 62(1), 1–9 (2002)
3. Gustafsson, J., Altenbernd, P., Ermedahl, A., Lisper, B.: Approximate worst-case execution time analysis for early stage embedded systems development. In: Lee, S., Narasimhan, P. (eds.) SEUS 2009. LNCS, vol. 5860, pp. 308–319. Springer, Heidelberg (2009)
4. Holzer, A., Januzaj, V., Kugele, S.: Towards Resource Consumption-aware Programming. In: ICSEA, pp. 490–493 (2009)
5. Lattner, C., Adve, V.S.: Llvm: A compilation framework for lifelong program analysis & transformation. In: CGO, pp. 75–88 (2004)
6. Holzer, A., Schallhart, C., Tautschnig, M., Veith, H.: How did you specify your test suite? In: ASE 2010 (to appear, 2010)
7. Holzer, A., Schallhart, C., Tautschnig, M., Veith, H.: Query-Driven Program Testing. In: Jones, N.D., Müller-Olm, M. (eds.) VMCAI 2009. LNCS, vol. 5403, pp. 151–166. Springer, Heidelberg (2009)
8. Holzer, A., Schallhart, C., Tautschnig, M., Veith, H.: FShell: Systematic Test Case Generation for Dynamic Analysis and Measurement. In: Gupta, A., Malik, S. (eds.) CAV 2008. LNCS, vol. 5123, pp. 209–213. Springer, Heidelberg (2008)
9. Giusto, P., Martin, G., Harcourt, E.: Reliable estimation of execution time of embedded software. In: DATE, pp. 580–589 (2001)
10. Januzaj, V., Mauersberger, R., Biechele, F.: Performance Modelling for Avionics Systems. In: Moreno-Díaz, R., Pichler, F., Quesada-Arencibia, A. (eds.) Computer Aided Systems Theory - EUROCAST 2009. LNCS, vol. 5717, pp. 833–840. Springer, Heidelberg (2009)
11. Mälardalen WCET research group: WCET Benchmarks (2010), http://www.mrtc.mdh.se/projects/wcet/benchmarks.html
12. Standard Performance Evaluation Corporation: SPEC CPU2006 (2010), http://www.spec.org/
13. Wilhelm, R., Engblom, J., Ermedahl, A., Holsti, N., Thesing, S., Whalley, D.B., Bernat, G., Ferdinand, C., Heckmann, R., Mitra, T., Mueller, F., Puaut, I., Puschner, P.P., Staschulat, J., Stenström, P.: The worst-case execution-time problem - overview of methods and survey of tools. ACM Trans. Embedded Comput. Syst. 7(3) (2008)

14. Li, Y.T.S., Malik, S.: Performance analysis of embedded software using implicit path enumeration. In: DAC, pp. 456–461 (1995)
15. Puschner, P., Schedl, A.: Computing maximum task execution times – a graph-based approach. Journal of Real-Time Systems 13(1), 67–91 (1997)
16. Ottosson, G., Sjödin, M.: Worst case execution time analysis for modern hardware architectures. In: SIGPLAN, pp. 47–55 (1997)
17. Lindgren, M., Hansson, H., Thane, H.: Using measurements to derive the worst-case execution time. In: RTCSA, pp. 15–22 (2000)
18. Smith, C.U., Woodside, M.: Performance validation at early stages of software development. In: System Performance Evaluation: Methodologies and Applications, pp. 383–396 (2000)
19. Mohanty, S., Prasanna, V.K., Neema, S., Davis, J.R.: Rapid design space exploration of heterogeneous embedded systems using symbolic search and multi-granular simulation. In: LCTES-SCOPES, pp. 18–27 (2002)
20. Nenova, S., Kästner, D.: Worst-case timing estimation and architecture exploration in early design phases. In: WCET (2009)
21. Wuerges, E., dos Santos, L.C.V., Furtado, O.J.V., Rigo, S.: An early real-time checker for retargetable compile-time analysis. In: SBCCI (2009)
22. Gustafsson, J., Ermedahl, A., Lisper, B., Sandberg, C., Källberg, L.: Alf - a language for wcet flow analysis. In: WCET (2009)
23. Wang, Z., Sanchez, A., Herkersdorf, A.: Scisim: a software performance estimation framework using source code instrumentation. In: WOSP, pp. 33–42 (2008)

Compiler-Support for Robust Multi-core Computing*

Raimund Kirner, Stephan Herhut, and Sven-Bodo Scholz

Department of Computer Science, University of Hertfordshire, Hatfield, United Kingdom
{r.kirner,s.a.herhut,s.scholz}@herts.ac.uk

Abstract. Embedded computing is characterised by the limited availability of computing resources. Further, embedded systems are often used in safety-critical applications with real-time constraints. Thus, the software development has to follow rigorous procedures to minimise the risk of system failures. However, besides the inherent application complexities, there is also an increased technology-based complexity due to the shift to concurrent programming of multi-core systems. For such systems it is quite challenging to develop safe and resource-efficient systems.

In this paper we give a plea for the need of better software development tools to cope with this challenge. For example, we outline how compilers can help to simplify the writing of fault-tolerant and robust software, which keeps the application code more compact, comprehensive, and maintainable. We take a rather extreme stand by promoting a functional programming approach. This functional programming paradigm reduces the complexity of program analysis and thus allows for more efficient and powerful techniques. We will implement an almost transparent support for robustness within the SAC research compiler, which accepts a C-like functional program as input. Compared to conventional approaches in the field of automatic software-controlled resilience, our functional setting will allow for lower overhead, making the approach interesting for embedded computing as well as for high-performance computing.

1 Introduction

In embedded computing it is important to have a high utilisation of resources, as resources tend to be limited. Furthermore, in safety-critical computing it is quite important to provide fault-tolerance for the most severe sources of faults. The classical term of fault tolerance implies that one has to built a fault model for the faults to be tolerated [1]. However, with increasing system complexity it has come to the point where it is very hard to identify the severe faults on a real physical system. As a result, the way to deal with this situation is to make systems robust, basically by enriching them with behaviour patterns that are believed to increase the likelihood of service-sustainability of the system [2]. Actually, robustness is a term still in the beginning of its solid definition, as it needs more research for behaviour patterns of a system that increase its robustness.

* The research leading to these results has received funding from the IST FP-7 research project "Asynchronous and Dynamic Virtualization through performance ANalysis to support Concurrency Engineering (ADVANCE)".

T. Margaria and B. Steffen (Eds.): ISoLA 2010, Part I, LNCS 6415, pp. 47–57, 2010.

Besides inclusion of robustness into the system design, it is also mandatory to follow a stringent engineering approach that minimises the risk of unintended system behaviour. For example, in the avionics domain the de-facto standard DO-178b [3] is used and in the automotive domain there is currently an active development of the ISO 26262 [4] standard as a guide for development of embedded software.

With the increasing concurrency within systems, it is also important to improve software development tools so provide a sufficient level of assistance to the developer. For example, in future embedded computing we have the challenge of providing robustness as well as efficient use of multi-core processors.

Our approach to tackle this problem is to develop a compiler that provides support for software-controlled robustness on multi-core architectures. There has been a lot of research in the area of compiler support for fault-recovery, especially in the field of high-performance computing [5,6,7]. However, there are a lot of technical issues like the overhead of checkpointing and the complexity of recovery.

Our approach is to use the functional programming paradigm, which simplifies many of the issues of fault-recovery, as the state of a component can always be narrowed down to the input data of this component, which allows for an efficient fine-grained recovery strategy. As a concrete compiler framework we use SAC (Single Assignment C), which is a functional programming language that has a syntax quite close to ANSI C [8,9]. SAC has a special strength on array data structures that allows a compiler to automatically generate concurrent code for that. Technically, SAC compiles to ANSI C code together with library calls to manage the concurrency. Thus, SAC is also a good choice for portability to embedded platforms, even though it has been originally developed for high-performance computing machines.

2 Robustness in Embedded Computing

Embedded systems are often used in a safety-critical environment where the correct behaviour of the system is of utmost importance. To achieve dependability, the classic approach is to enrich the system with mechanisms that allow to tolerate a set of explicitly described faults. The prerequisite to design fault-tolerant systems is the definition of a fault model and the intended level of tolerance. Though quite useful as a design concept, fault-tolerant computing is only as good as the level of realism of the fault model is. For example, slightly-off-specification faults can be quite subtle and it is very hard to completely prerecognize them [10]. With this realisation, the focus in dependable computing is currently shifting from fault-tolerance to robustness, where the basic idea of a system to be robust is to maintain an acceptable level of service despite various *unexpected* perturbations [11,2]. The origin of unexpected perturbations can be manifold, as described by Obermaisser and Kopetz:

> *"Robustness comprises the provision of an acceptable level of service despite the occurrence of transient and permanent hardware faults, design faults, imprecise specifications, and accidental operational faults."* [12]

A robust system is expected to maintain operation but may reduce the level of service during operation, even producing less accurate but still useful output. Robustness is a

system property that is inherently hard to evaluate, as it by definition deals with *unexpected* perturbations. However, there are behaviour patterns that are believed to make a system robust. Among these patterns are resilience, adaptability, recovery, recursive restartability, repairability, anytime computation, or degeneracy [2]. Note that these patterns are not strictly orthogonal, some of them even enforce other.

resilience is the system's ability to compensate for a temporal degradation of the provided level of service.

adaptability is the system's ability to change its behaviour, or internal or external structure in order to compensate for perturbations or changing requirements.

recoverability is the system's ability to detect and remove errors by restoring a correct state of the affected component. Recoverability avoids the accumulation of errors within the system state.

recursive restartability is the system's ability to restart small parts of the system independently in order to remove any erroneous state in these parts after a transient fault. A partial restart is expected to cause less delay and service disruption than a full restart. The "recursiveness" comes from the subdivision of a system into subsystems, where a restart of a (sub)system will require to also restart all its nested subsystems.

repairability is the system's provision of an interface and mechanisms that allow to repair a system after a transient or permanent fault has occurred. After the repair of a system component, the system has to be able to reintegrate the component and its providing services into the system.

anytime computation is an approximating implementation style where the system iteratively refines the result. The benefit of anytime computation is that the intermediate result can be used as an approximation of the final result in case that the system gets short on computational resources [13,14].

degeneracy is the degree to which a part of the system can take over the functionality of another part of the system. Degeneracy is different from redundancy, as redundancy is meant to add extra resources for the provision of the same service, while degeneracy focuses on the ability of changing the use of a resource by reusing it for the provision of a different service.

In fact, above list of behaviour patterns is not meant to be exhaustive, nor does it mean that a system has to implement all of them to be sufficiently robust, as it is always a matter of application context to decide what level of robustness is sufficient. And of course, it is impossible to implement an absolutely robust system that can withstand whatever kind of perturbations will come.

3 Compiler Support for Robustness

In the following we discuss how development tools like a compiler can support the system developer in developing robust concurrent applications.

There are certain robustness patterns that are best achieved by explicitly programming them at the application level. For example, to deploy the *anytime computation* [13,14] patterns one has to choose appropriate algorithms that are suitable for

anytime computation. Translating a program automatically into anytime computation has not been done so far and it is also questionable whether an automatic translation would result in efficient resource usage. Further, anytime computation is not meant to be applied for individual instructions. Instead the whole algorithm should be trimmed to anytime computation. Thus, anytime computation so far is not considered as a subject for compiler support.

The *adaptability* pattern is also tightly linked to the concrete application semantics. However, the adaptability behaviour is still somehow orthogonal to the logic operations performed by the application. Thus, it would make sense to program adaptability behaviour in a coordination language like S-Net [15,16]. We are currently working on adding such support for robustness into the S-Net compiler. As some first results we derived a specialised variant of S-Net, which is resource-boundable [17]. To perform system adaptation it is first also necessary to be able to detect the errors to react on. However, this does not necessarily have to be a reactive process. There exists also research towards automatic proactive fault detection [18,19,20].

Resilience is an emerging behaviour pattern that is driven by others. For example, *resilience* can be realised based on *recovery*. Given that the system state is saved on some regular basis, whenever an error is detected a recovery can be evoked by restoring the last saved correct system state [21,22]. Depending on the type of fault and the system architecture, it might be sufficient to restore the state only partially, which would be more efficient than a global state restoration [23]. However, such a selective recovery is more complex to realise. If there is a permanent fault, a reconfiguration might be necessary for proceeding the operation. Till having finished the recovery process the system's level of service may temporarily degrade. The placement of checkpoint code and recovery code can be controlled by the compiler, as it has a relatively rich knowledge about the program code.

Whenever there is a faulty local state detected in part of the system, *resilience* might be also achieved by restarting this part of the system, thus bringing it back to a correct initial state. In general, restarting is a more simple technique than recovery as it does not require to regularly store the system state for recovery. However, in complex calculations a recovery might be more beneficial as a restart, as the restart will cause to loose any intermediate result having been calculated so far by this part of the system. The compiler support for recursive system restart is similar to the second part of system recovery, the detection of erroneous states and their removal.

The *Repairability* pattern makes the overall design choices to achieve robustness explicit. Accepting that it is not always possible to avoid all faults or to mask them, one has to provide efficient solutions to regain a correct system state. A repair action typically consists of the following actions [2]:

1. Error detection
2. Location of the erroneous part of the system
3. Analysis to direct the repair action
4. Repair action
5. Reintegration of the service

The specific repair action can be one of the already discussed measures like restart, rollback, etc. It is important that the runtime system provides the support for the

reintegration of the repaired component. This reintegration might be an issue for the compiler is case that the reintegration is done at the level of a runtime layer that is generated by the compiler.

Degeneracy is an import design criterion of robustness, that also offers a potential for strong benefit from compiler support. The compiler can generate the code of a service implementation for different platforms such that the runtime system can migrate a service to a different hardware component. In less resource-constrained systems the recompilation for a different hardware itself may be also performed during runtime, adding further flexibility to this robustness pattern.

4 Robustness in a Functional Setting

As detailed in the previous section, we consider recoverability, recursive restartability and degeneracy as the key aspects of robustness that should be tackled, at least partially, at the compiler level. All three of these require that a thread of execution can be stopped and its state captured. In the context of recoverability, it suffices to store the state of the entire system or application, commonly referred to as checkpointing, such that the system can be resumed from there later. However, the main overhead in checkpointing in modern systems stems from storing the associated state [21]. Therefore it is beneficial to minimise the state that is to be stored. Rich type systems as they are commonly found in functional programming languages can be of great value in this context. By exploiting structural information on heap objects derived from their types, the size of the checkpoint data can easily be reduced by up to 70% [22].

Rich type information becomes even more valuable in the context of recursive restartability. Here, a global checkpoint does not suffice, as this would only allow to restart the entire computation. Instead, checkpoints for each sub-computation need to be generated and stored. Reducing checkpoint size is therefore even more important. Another key aspect in this setting, however, is to compute the containment of a sub computation, i.e., those parts of the global state that are required to restart the computation and those that are modified by the computation. Again, functional programming languages have key advantages here. Due to their call-by-value semantics, which ensure that arguments to functions are immutable, and explicit modelling of side-effects, containment of function state is well defined in a functional setting. Only the arguments to a function are required to restart its computation and the function itself does not modify any global state. Therefore, support for reversing and repeating a computation in a compiler for functional languages is simple to implement compared to the imperative setting [24].

Lastly, rich type information and the strict containment of function state can be exploited in the context of degeneration, as well. A key requirement to be able to tolerate the failure of a hardware component or subsystem is the ability to migrate the program and its state from the failed component to another component of the system. In today's heterogeneous systems, this migration might involve, apart from capturing the state, a conversion of the data. In weakly typed imperative settings like C, migrating state across platforms requires sophisticated compiler support and involves significant runtime costs [25]. The two key challenges involved, capturing the state of a computation and analysing the data-layout, are greatly simplified in a functional setting. We

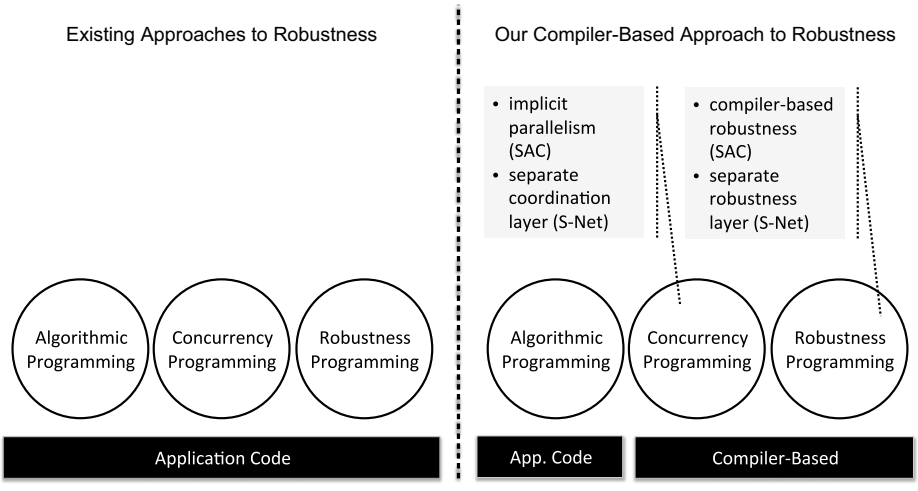


Fig. 1. Overview of our approach to compiler based robustness versus classical approaches

have argued for the former before. The latter, data-layout analysis, is supported greatly by the rich type annotations. It is common practise in functional languages to generate boiler-plate code like serialisers and representation transformations automatically [26].

A general overview of our approach and the divide between programmer supported tasks and compiler supported tasks in robust concurrent programming is given in Figure 1. In classical approaches, as shown on the left-hand side, the full responsibility lies with the application programmer. In the source code the arithmetic programming is typically mixed up with concurrency programming and, if needed, also robustness programming. We shift the responsibility for concurrency and robustness, at least in part, to the compiler, as shown on the right-hand side of Figure 1. We see concurrency programming ideally as an implicit activity to simplify concerns, or even separate it by using a coordination language like S-Net for it. Robustness programming should also be kept separate from the algorithmic programming of the application. Robustness programming would also fit quite will into the concept of a coordination language like S-Net. Further, we see a significant contribution of robustness programming can be done automatically by the compiler, with optional guidance by the developer.

4.1 SAC - Data-Parallel Functional Programming

To investigate the techniques and key challenges in compiler support for robustness in the setting of functional languages, we have chosen to use Single Assignment C [27,8] and its compiler[1] as test environment.

Single Assignment C, or SAC for short, is a first-order applicative functional programming language designed initially with high-performance and high-productivity

[1] The compiler is available for download from the project's website at `http://www.sac-home.org`

programming in mind. It has a syntax similar to C, combined with a programming model that resembles that of MATLAB.

The focus on high programmer productivity shows on two levels in SAC. Firstly, SAC programs are specified using a rather high level of abstraction compared to classical high-performance languages in the imperative domain. To allow for this, we have developed compiler technology that maps programs specified at high levels of abstraction to efficient low-level implementations on various architectures.

Secondly, those complexities of programming that do not contribute to the actual algorithm design, i.e., memory management, concurrency and scheduling, are handled implicitly by the compiler, as opposed to being done by the programmer like in most other languages. This, on the one hand, simplifies program specification and reduces the likelihood of programming errors. On the other hand, and even more importantly in the context of compiler supported robustness, it gives the compiler full control over the actual realisation of those aspects.

Although SAC was designed for large-scale high-performance applications, many of its features prove beneficial in the embedded domain, as well. In large-scale high-performance applications, efficient use of resources is of similar importance as in embedded systems. For instance, the memory footprint needs to be minimised to ensure that applications can run within existing resource bounds. Using deferred heap management by means of garbage collection, which is predominantly used in functional programming languages, therefore is not an option in SAC. To minimise memory usage, frequent garbage collection phases would be required. Those, however, have a detrimental effect on program runtime. SAC therefore uses non-deferred heap management via reference counting. Using reference counting, the size of the heap can be constantly kept at a minimum without periodically disrupting program execution. Instead, reference counting operations are explicitly inserted throughout the program code, leading to a predictable runtime cost. The latter property of reference counting, i.e., its predictable runtime behaviour and cost, is also important in the context of real-time systems.

As mentioned before, functional programming languages in general use call-by-value semantics, i.e., state is not shared across function boundaries but instead arguments are copied. However, for performance reasons, this property is often weakened in practise. For larger data-structures like arrays, where copying comes at a significant runtime cost, references and mutable data-structures are used instead. This, however, impedes static analyses and reasoning on programs. In SAC, all data-structures are immutable on the language level. Thus, once defined, their value cannot change regardless of the context they are passed into. Updating a value, at least conceptually, always produces a fresh copy. This rather rigorous setting allows us to apply advanced program rewriting techniques. Even more importantly, it gives the SAC compiler ultimate freedom to place objects in memory, distribute them and schedule their computation, as different threads of computation cannot share state but merely values. Of course, to achieve good runtime performance, at runtime mutable objects are used. However, the important distinction here is that these are introduced by the compiler as opposed to by the programmer.

Finally, the last feature that is of particular importance in the context of compiler support for robustness is the implicit nature of concurrency in SAC. By design, SAC puts

forward a programming model that supports the programmer in expressing algorithms such that they naturally contain implicitly parallel operations. These are then mapped by the SAC compiler to the concurrency resources of the target platform. The mapping thereby can make use from large degrees of freedom, allowing the compiler to compute a range of schedules, from fully sequential to fine-grained massively parallel execution. Combined with the platform independent nature of SAC, this allows us to generate code for a range of systems from standard platforms like symmetric multi-processors to very restricted targets like GPGPUs.

4.2 Support for Robustness with SAC

Our roadmap for adding robustness support to the SAC research compiler currently lists three major sub projects. Firstly, we will explore different fault detection techniques like software-based replication [6] and timeout-based failure detection. For the former, we are particularly interested to find what advantages a functional setting offers with respect to replicated execution. Apart from the simplified separation of state, we expect easier code generation for checking the validity of states in a coarser grained fashion.

Supplying interfaces to the underlying operating system and hardware to cater for proactive fault detection would be an interesting addition here. However, we do not expect an advantage from the functional setting in this case and will thus leave this as an optional future addition.

With this concept we hope to address single-event upsets as well as permanent local faults of hardware components. However, a thorough investigation of possible fault sources and techniques how to detect or tolerate them is another crucial component of this step.

Once we have the techniques in place to detect faults, a next step will be to adapt code generation such that faults can be tolerated and program execution can recover. In this setting, we plan to focus on adapting the scheduling of concurrent tasks in SAC to a robust setting. As SAC features fully implicit memory management and concurrency, we have a tight grip on the actual implementation of these by the compiler. Other than in C, for instance, where the communication pattern and scheduling is usually hard-wired using POSIX threads or MPI, in SAC the compiler has more freedom. If the failure of one computation is detected, the computation can be restarted. Here, the functional nature of SAC and the use of immutable data structures in SAC turns out to be very beneficial. By using our existing liveness analysis for reference counting, we can easily detect the active heap objects that take part in a computation. Furthermore, functions and parallel operations seem to be a natural guide for the granularity of recovery. As found previously in the context of ZPL [28], checkpoints are best placed in between parallel sections of execution. These optimal code positions are easily identifiable in SAC. Other than the previous work, however, in SAC we will implement recovery using a much finer granularity. Due to the side-effect free nature of parallel computations in SAC, we are able to restart each thread of computation and recover on the level of partial results. For this, we borrow techniques originally developed in the setting of software transactual memory for functional languages [29].

Lastly, we will investigate support for degeneracy in the SAC compiler. Our current compiler already features many of the required techniques. For instance, due to

the high-level nature of SAC, we are able to generate code for a variety of platforms from a single source specification. Thus, we can support truly portable migration by using the compiler alone. As an example, consider a setup with a legacy computing node equipped with a special purpose co-processor like a GPU. Initially, we would emit code for the special purpose processor to maximise throughput and reduce energy costs. However, if such processor fails, we could automatically fall back to a generic implementation on the legacy node.

We are currently extending this approach to runtime adaptive codes [30]. The idea here is to recompile the program at runtime to adapt to a changing environment. Our current implementation focuses on adapting to changes in the input data, however this could be extended to fault-tolerance scenarios, as well.

5 Discussion

One might wonder whether the functional paradigm has any potential impact in the field of embedded computing. But, for example, with the programming language HUME there exist already well-suited functional programming environments for the embedded domain [31, 32, 33]. Furthermore, in safety-critical computing, people use code guidelines like MISRA [34] in the automotive domain, that provide rules that restrict conventional imperative programming languages almost to functional programming patterns. Thus, whenever it comes to safety-critical systems, the functional computing paradigm has a realistic impact potential.

The approach presented so far just highlights the potential for software-controlled robustness. However, we have already identified some special benefits of deploying a functional setting for software-controlled robustness. Exploiting the rich static knowledge about resource usage and the structure of a system's state, combined with the side-effect free nature of computation in the functional setting, has the potential to find more simple and efficient solutions as have been developed so far for conventional imperative programming paradigms.

We think that increased tool support for software-controlled robustness and multi-core deployment will become essential for the development of future embedded applications in the avionic and automotive domain. With the current rather inflexible development approaches it will become increasingly challenging to maintain efficiency and robust behaviour for future development platforms.

References

1. Avižienis, A., Laprie, J.C., Randell, B., Landwehr, C.: Basic concepts and taxonomy of dependable and secure computing. IEEE Transactions on Dependable and Secure Computing 1(1), 11–33 (2004)
2. Mikolasek, V.: Dependability and robustness: State of the art and challenges. In: Proc. Workshop on Software Technologies for Future Dependable Distributed Systems, Tokyo, Japan (March 2009)
3. RTCA: Software considerations in airborne systems and equipment certification. RTCA/DO-178B (1992)

4. ISO/DIS: Road vehicles – functional safety. ISO/DIS standard 26262
5. Treaster, M.: A survey of fault-tolerance and fault-recovery techniques in parallel systems. ACM Computing Research Repository (CoRR) abs/cs/0501002 (2005)
6. Reis, G.A., Chang, J., Vachharajani, N., Rangan, R., August, D.I.: SWIFT: Software implemented fault tolerance. In: Proc. 3rd International Symposium on Code Generation and Optimization (CGO) (March 2005)
7. Chang, J., Reis, G.A., August, D.I.: Automatic instruction-level software-only recovery. IEEE Micro 27(1), 36–47 (2007)
8. Grelck, C., Scholz, S.B.: SAC: A functional array language for efficient multithreaded execution. International Journal of Parallel Programming 34(4), 383–427 (2006)
9. Grelck, C.: Shared memory multiprocessor support for functional array processing in SAC. Journal of Functional Programming 15(3), 353–401 (2005)
10. Ademaj, A.: Slightly-off-specification failures in the time-triggered architecture. In: Proc. 7th IEEE International Workshop on High Level Design Validation and Test, Cannes, France, pp. 7–12 (October 2002)
11. Mikolasek, V.: Robustness in complex systems - state of the art report. Research Report 26/2008, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 1-3/182-1, 1040 Vienna, Austria (2008)
12. Obermaisser, R., Kopetz, H.: From ARTEMIS requirements to a cross-domain embedded system architecture. In: Proc. Embedded Real Time Software and Systems, Toulouse, France (May 2010)
13. Horvitz, E.J.: Reasoning about beliefs and actions under computation resource constraints. In: Proc. Workshop on Uncertainty in Artificial Intelligence, Seattle, Washington (1987)
14. Boddy, M., Dean, T.: Solving time-dependent planning problems. In: Proc. 11th International Joint Conference on Artificial Intelligence (August 1989)
15. Grelck, C., Scholz, S.B., Shafarenko, A.: A Gentle Introduction to S-Net: Typed Stream Processing and Declarative Coordination of Asynchronous Components. Parallel Processing Letters 18(2), 221–237 (2008)
16. Shafarenko, A., Scholz, S.B., Grelck, C.: Streaming networks for coordinating data-parallel programs. In: Virbitskaite, I., Voronkov, A. (eds.) PSI 2006. LNCS, vol. 4378, pp. 451–455. Springer, Heidelberg (2007)
17. Kirner, R., Scholz, S.B., Penczek, F., Shafarenko, A.: PS-NET - a predictable typed coordination language for stream processing in resource-constrained environments. In: Proc. 1st Int'l Conference on Computational Logics, Algebras, Programming, Tools, and Benchmarking (submitted, November 2010)
18. Vallee, G., Engelmann, C., Tikotekar, A., Naughton, T., Charoenpornwattana, K., Leangsuksun, C., Scott, S.L.: A framework for proactive fault tolerance. In: Proc. 3rd Int'l Converence of Availability, Reliability and Security, Barcelona, Spain, pp. 659–664 (May 2008)
19. Lee, C., Lee, D., Koo, J., Chung, J.: Proactive fault detection schema for enterprise information system using statistical process control. In: Proc. Conference on Symposium on Human Interface 2009, pp. 113–122. Springer, Heidelberg (2009)
20. Wang, C., Mueller, F., Engelmann, C., Scott, S.L.: Proactive process-level live migration in hpc environments. In: Proc. ACM/IEEE conference on Supercomputing (SC 2008), Piscataway, NJ, USA. IEEE Press, Los Alamitos (2008)
21. Elnozahy, E.N.M., Alvisi, L., Wang, Y.M., Johnson, D.B.: A survey of rollback-recovery protocols in message-passing systems. ACM Compututing Surveys 34(3), 375–408 (2002)
22. Choi, S.E., Deitz, S.J.: Compiler support for automatic checkpointing. In: Proc. 16th Annual International Symposium on High Performance Computing Systems and Applications, Washington, DC, USA, p. 213. IEEE Computer Society, Los Alamitos (2002)

23. Dinan, J., Singri, A., Sadayappan, P., Krishnamoorthy, S.: Selective recovery from failures in a task parallel programming model. In: Proc. IEEE International Symposium on Cluster Computing and the Grid, pp. 709–714. IEEE Computer Society, Los Alamitos (2010)
24. Harris, T., Marlow, S., Peyton-Jones, S., Herlihy, M.: Composable memory transactions. In: Proc. 10th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, pp. 48–60. ACM, New York (2005)
25. Ramkumar, B., Strumpen, V.: Portable checkpointing for heterogeneous archtitectures. In: Proc. 27th International Symposium on Fault-Tolerant Computing (FTCS 1997), Washington, DC, USA, p. 58. IEEE Computer Society, Los Alamitos (1997)
26. Hinze, R.: A new approach to generic functional programming. In: Proc. 27th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pp. 119–132. ACM Press, New York (2000)
27. Scholz, S.B.: Single Assignment C — efficient support for high-level array operations in a functional setting. Journal of Functional Programming 13(6), 1005–1059 (2003)
28. Choi, S.E., Deitz, S.J.: Compiler support for automatic checkpointing. In: HPCS 2002: Proceedings of the 16th Annual International Symposium on High Performance Computing Systems and Applications, Washington, DC, USA, p. 213. IEEE Computer Society, Los Alamitos (2002)
29. Harris, T., Marlow, S., Peyton-Jones, S., Herlihy, M.: Composable memory transactions. In: PPoPP 2005: Proceedings of the Tenth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, pp. 48–60. ACM, New York (2005)
30. Grelck, C., van Deurzen, T., Herhut, S., Scholz, S.B.: An Adaptive Compilation Framework for Generic Data-Parallel Array Programming. In: 15th Workshop on Compilers for Parallel Computing (CPC 2010), Vienna University of Technology, Vienna, Austria (2010)
31. Patai, G., Hanák, P.: Embedded functional programming in Hume. In: IASTED on Software Engineering, Innsbruck, Austria, pp. 328–333. ACTA Press (2007)
32. Hammond, K., Michaelson, G.: The design of Hume: A high-level language for the real-time embedded systems domain. In: Lengauer, C., Batory, D., Consel, C., Odersky, M. (eds.) Domain-Specific Program Generation. LNCS, vol. 3016, pp. 127–142. Springer, Heidelberg (2004)
33. Hammond, K., Michaelson, G.: Hume: A domain-specific language for real-time embedded systems. In: Pfenning, F., Smaragdakis, Y. (eds.) GPCE 2003. LNCS, vol. 2830, pp. 37–56. Springer, Heidelberg (2003)
34. MISRA, T.M.I.S.R.A.: MISRA-C 2004: Guidelines for the Use of the C Language in Critical Systems. MISRA (October 2004), ISBN: 0-9524156-4-X (pdf version)

Thematic Track: Formal Languages and Methods for Designing and Verifying Complex Embedded Systems

Yamine Ait Ameur[1], Frédéric Boniol[2], Dominique Mery[3], and Virginie Wiels[2]

[1]LISI/ENSMA, Poitiers France
amine@ensma.fr
[2]ONERA/DTIM, University of Toulouse, France
firstname.name@onera.fr
[3]LORIA, University of Nancy, France
mery@loria.fr

Nowadays, it is well accepted that the develop- ment of critical systems involves the use of formal methods. One of the major fields where these methods made a lot of progress are the avionics, aerospace, transport areas, telecom, etc. These systems are responsible for various functions, such as navigation, guidance, stability, power management, board/ground communications, passenger entertainment.... Moreover, their complexity is continuously growing.

Due to safety constraints, these systems often have to go through certification. This requires testing, and a design process based on a set of tight rules. However, due to the increasing complexity of described systems, there is clearly no guarantee that such tight rules and rigorous testing will lead to error free systems. An alternative approach for helping system designers is formal methods, i.e. fundamental languages, techniques and tools for design, analysis, validation or transformation of systems in a provably correct way.

Indeed, formal techniques, in particular formal specification languages and associated proof tools, could be an advantageous alternative or at least a good complement which would facilitate a significant reduction in test phases. Several formal languages methods, tools and techniques have been applied for the development of such systems in different parts of the world and they have been put into practice during the development of actual, specific programs (aircraft, space vehicle...).

This thematic track is devoted to compile the state-of-the-art in formal methods applied to the development of embedded systems. It will highlight on the recent advances in the use of these methods.

The four papers of the "*Formal Languages and Methods for Designing and Verifying complex Engineering Systems*" track are representative of the ISOLA ambition.

Two papers concern application of formal methods in security. "*Analyzing the security in the GSM radio network using attack jungles*" proposes the concept of attack jungles to formalize vulnerabilities of systems and a method to analyze

T. Margaria and B. Steffen (Eds.): ISoLA 2010, Part I, LNCS 6415, pp. 58–59, 2010.

security attacks. "*Formal modeling and verification of sensor network encryption protocol in the OTS/CafeOBJ method*" describes the formal verification of security properties of an encryption protocol using an algebraic framework.

Two papers provide necessary ingredients for applications to be possible: tools and leveraging techniques. "*Model-Driven Design-Space Exploration for Embedded Systems: The Octopus Toolset*" presents a toolset that facilitates reuse of models and combination of different analyzes for design space exploration of embedded systems. "*Contract-based slicing*" proposes a technique for slicing programs using contract-based annotations included in these programs.

Analyzing the Security in the GSM Radio Network Using Attack Jungles

Parosh Aziz Abdulla[1], Jonathan Cederberg[1], and Lisa Kaati[2,⋆]

[1] University of Uppsala, Sweden
{parosh,jonathan.cederberg}@it.uu.se
[2] FOI Swedish Defence Research Agency, Sweden
lisa.kaati@foi.se

Abstract. In this paper we introduce the concept of attack jungles, which is a formalism for systematic representation of the vulnerabilities of systems. An attack jungle is a graph representation of all ways in which an attacker successfully can achieve his goal. Attack jungles are an extension of attack trees [13] that allows multiple roots, cycles and reusability of resources. We have implemented a prototype tool for constructing and analyzing attack jungles. The tool was used to analyze the security of the GSM (radio) access network.

1 Introduction

Analyzing the security of complex systems is a difficult but important task. In industrial projects, security analysis is usually performed by a team of experts that create a model of the system and then analyze the security manually. The drawback of this approach is that, for complex systems, the model grows to a size that no longer permits manual analysis, thus creating a need for tools that partially automate the process.

In collaboration with a team of security experts at Ericsson Research the security of the GSM radio network was analyzed. To do this, we used a formalism called *attack trees* where the vulnerability of a system is modeled by finding all possible attacks towards a system. Attack trees were first introduced by Bruce Schneier [13] and later formalized by Mauw and Oostdijk in [11]. An attack tree is an acyclic graph with one root node. The root node represents the global goal and the tree itself encodes all attacks that are possible on the system. There are two types of nodes in an attack tree, and-nodes and or-nodes. Satisfying a node means either satisfying all predecessor nodes (and-node) or satisfying one predecessor node (or-node).

When using the formalism described in [11], we encountered problems regarding both the modeling and the analysis of attacks. One problem that occurred when modeling the attacks was the appearance of cycles. A problem that occurred when analyzing attacks was the fact that some nodes could, once they hade been accomplished, be re-used over and over again.

⋆ This work was supported by a grant from The Swedish Governmental Agency for Innovation Systems.

T. Margaria and B. Steffen (Eds.): ISoLA 2010, Part I, LNCS 6415, pp. 60–74, 2010.

To overcome these problems we have extended the notion of attack trees to *attack jungles*. There are three aspects that distinguish attack jungles from attack trees. First of all, an attack jungle may have several root nodes (an attack tree only has one root), where each root is a global goal of an attacker. Secondly, we introduce the possibility to use nodes that are *reusable*. A reusable node reflects the fact that the knowledge gained from a node can be used in several attacks. An example of reusability is obtaining a secret cryptographic key. If an attacker obtains a secret key, all messages encrypted using the key can also be decrypted by the attacker. Reusability is an important property since some resources only need to be exploited once but can affect several of the identified threats. Finally, we allow an attack jungle to contain cycles. Cycles are useful when modeling attacks that depend on each other. One example is if you have an address you can easily find the name of the person living at the address and vice versa.

Once the attack jungle is created, it can be analyzed to gain information about the security of the system. The analysis is done using an algorithm that relies on the principle of backward reachability analysis.

The analysis of an an attack jungle can answer a number of different questions about the attack jungle, such as "Is any attack possible?", "What is the minimum cost of performing an attack?" and "What is the minimum required skill level for an attack?"

The formal representation of an attack jungle enables tools to both identify and analyze possible threats to a specific system. Good tool support is required since attack jungles can become large and complex (a full attack jungle may contain thousands of different paths all leading to completion of the attack).

We have implemented a prototype tool that can be used to model and analyze attack jungles. The tool was used in a study conducted to analyze the security of the GSM radio access network where attacks towards the network were both modeled and analyzed. For attacks that had a high risk of occurring, countermeasures were designed and the tool was used to evaluate the impact/benefit of implementing the different countermeasures.

1.1 Related Work

Attack trees was introduced by Schneier [13] and later formalized by Mauw and Oostdijk in [11] where the authors provide a framework for attack trees including algorithms to analyze the attack trees.

Defence trees [5] are an extension of attack trees with attack countermeasures. In a defence tree each leaf is decorated with a set of countermeasures. Each countermeasure represents a possible risk relaxation when the specific vulnerability is used. In [4] defence trees are used to represent attack scenarios and game theory to analyze the possible strategies of the attacker and the defender (or system administrator). In [6] defence trees that are enriched with conditional preferences are introduced. These structure called are called CP-defence trees.

Attack trees are not only used to analyze the security of existing computer systems. A similar formalism called obstacle trees [9] are used when caring for security at requirements engineering time. Obstacle trees are a part of a

goal-oriented framework, where anti-goals that threatens the security of the system are setup by an attacker [15]. Another way to use attack trees are described in [7] were they are used as a tool by intelligence analysts and decision-makers in their work.

Another approach to analyze the security of a system is to use a formalism with a similar name called attack graphs [14] [12]. Attack graphs are automatically generated from a model of the system and the attacks towards the system are found automatically using model checking techniques.This approach is completely different from ours since the attacks themselves are not modeled, instead the system is modeled and then the attacks are found automatically.

There are some commercial tools that use attack trees. One is AttackTree+ by isograph [3] and another is SecurITree by Amenaza Technologies [10]. These tools do not allow the attack tree to contain cycles and the concept of reusability of resources can not be modeled.

1.2 Outline

In the next section we introduce the concept of an attack jungle. Section 3 presents an algorithm used for analyzing an attack jungle based on backward reachability analysis. Section 4 describes how to use the result of the analysis to automatically derive certain attributes of the modeled system. Section 5 contains a case study of the security in the GSM network, we give a short description of the different components in the GSM network and describe some of the threats towards the GSM radio access network that was identified. Finally, in section 6 we give some concluding remarks and some directions for future work.

2 The Attack Jungle Formalism

In this section we give some preliminaries of *Attack Jungles*. The presentation follows that of [11] in many respects, but we have reformulated some concepts to fit our more general formalism.

For a set A, we use $\mathcal{M}(A)$ to denote the set of multisets over A. For a multiset $m \in \mathcal{M}(A)$, we write $m(a)$ to denote the number of occurrences of a in m. Sometimes, we write multisets as lists, so $[a, a, b]$ denotes the multiset with two occurrences of a and one of b (i.e. $m(a) = 2$ and $m(b) = 1$). For a multiset m and an element a, we write $a \in m$ to denote that $m(a) > 0$. For multisets m_1 and m_2, we let $m_1 + m_2$ be the multiset such that $(m_1 + m_2)(a) = m_1(a) + m_2(a)$, and we let $m_1 - m_2$ be the multiset such that $(m_1 - m_2)(a) = \max(0, m_1(a) - m_2(a))$. For multisets m_1 and m_2, we use $m_1 \leq m_2$ to denote that $m_1(a) \leq m_2(a)$ for all a. We define $|m| := \sum_{a \in m} m(a)$ i.e., $|m|$ is the sum of the multiplicities of all elements in m.

We show a typical example of an attack jungle in Figure 1. An attack jungle is a graph with two types of nodes: and-nodes and or-nodes. An attack jungle encodes in a hierarchical manner a set of attacks on a specific target. More formally, an attack jungle is a multigraph $J = (V, E, A)$, where

- V is a set of nodes, each describing an attack goal.
- $E \in \mathcal{M}(V \times V)$ is a multiset of edges
- $A \subseteq V$ is a unary relation on the nodes, describing which nodes are and-nodes. The complement $V \setminus A$ a set of all or-nodes.

Note that some of the nodes in the attack jungle have no incoming edges. These nodes are called *attack components* and are the lowest level of abstraction that the model provides. The set of attack components is denoted by $\mathbb{C}$.

We illustrate how to interpret an attack jungle using the example of Figure 1. Consider the attack jungle in Figure 1. Each node in the graph describes an attack goal. Most such goals are not what we intuitively would think of as the goal of an attack, but rather some component (intermediate step) of an attack. For example, "get address" is not really an attack in itself, but a necessary component of an attack where something is to be sent to the victim. To realize the attack goal "buy bomb", the attacker has to "find seller" and "pay seller". The attack goal "find seller", in turn, can be accomplished either by a "search on Internet" or "ask friends". When realizing "buy bomb" we have to accomplish all attack goals below, but for "find seller" any one will suffice. This is because the former node is an and-node, whereas the latter is an or-node. This is illustrated in Figure 1 by the existence of a small arc between the incoming arrows of "buy bomb", whereas there is no such arc for "find seller".

Note that in the definition of the attack jungle, the set of edges is a multiset rather than an ordinary set, making the graph a multigraph rather than a standard graph. The reason for this is to capture situations where one needs multiple instances of a resource to accomplish a goal.

With this interpretation of an attack jungle, a question arises: Given an attack goal v, how does one identify all multisets of attack components from which we can accomplish v? To answer this question, we first need to make precise the notion of accomplishing an attack goal.

For an attack jungle (V, E, A), a *state* is a multiset $s \in \mathcal{M}(V)$. Being in a specific state, means that all the attack goals corresponding to the nodes in the state are accomplished. To formalize the notion of accomplishing an attack goal,

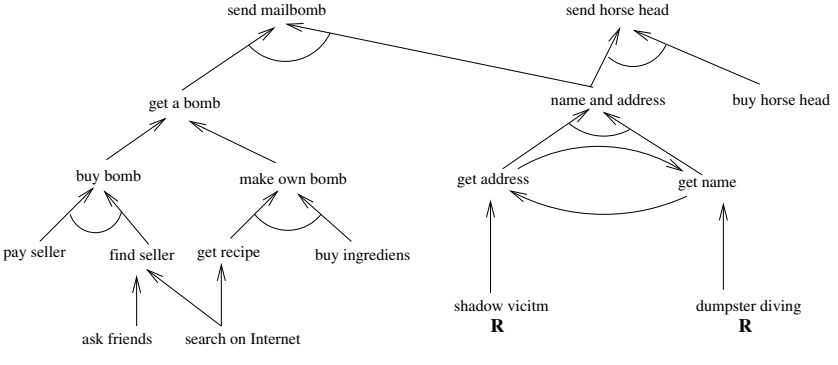


Fig. 1. An Attack Jungle

we define a transition relation on states that describes how a specific attack goal can be accomplished from other attack goals. We first define a relation $\Gamma \subseteq \mathcal{M}(V) \times V$ describing how we can accomplish a specific attack goal from its direct predecessors in the attack jungle. As described above, we need only accomplish one of the direct predecessors of an or-node, whereas all direct predecessors of an and-node have to be accomplished. More formally,

$$\Gamma(m, v) \text{ iff } \begin{cases} v \in A \text{ and } m(u) = E((u, v)) \text{ for all } u \in V \\ v \notin A, m = [u] \text{ for some } u \in V, \text{ and } (u, v) \in E \end{cases}$$

We are now ready to define a transition relation $\longrightarrow$ on states. For states s, s', we have that $s \longrightarrow s'$ iff there is a multiset $m \leq s$ such that $\Gamma(m, v)$ and $s' = (s - m) + [v]$.

The meaning of the transition relation is illustrated by the following example. If the attack jungle in question is the one depicted in Figure 1, we can make a transition from the state ["name and address", "make own bomb"] to the state ["name and address", "get a bomb"], and then to the state ["send mailbomb"].

Given this formalism, we can see that the problem of identifying sets of attack components from which we can accomplish a goal node $g \in V$ reduces to the problem of determining reachability in the above transition system. Formally, this amounts to finding the set $A \subseteq \mathcal{M}(\mathbb{C})$ such that for each $a \in A$, $a \stackrel{*}{\longrightarrow} G$ where $G \geq [g]$, i.e., the set of initial states in the transition system is $\mathcal{M}(\mathbb{C})$, and the set of final states is $\{G \in \mathcal{M}(\mathbb{C}) : G \geq [g]\}$ In the next section, we will show how to solve this problem.

3 Algorithm

To solve the problem described in the previous section, we will employ the technique of *Symbolic Backwards Reachability*. We first identify an interesting property of the transition system: its *monotonicity*.

A monotonic transition system has the property that for states s_1, s_2, s_3, such that $s_1 \longrightarrow s_3$ and $s_1 \leq s_2$, there is a state s_4, such that $s_2 \longrightarrow s_4$ and $s_3 \leq s_4$. The monotonicity of our transition system follows directly from its definition.

Monotonicity allows for efficient representation of sets of states through the concept of *upward-closed* sets of states, using the minimal element of such sets as representative. The upward closed set represented by a state s is the set $\{s' | s \leq s'\}$. For a monotonic transition system, upward-closedness is preserved by the transition relation. For more details on monotonic transition systems and their properties, see [2].

The idea behind backward reachability analysis is to start with the final state we want to reach, and then successively compute all states from which this target is reachable. If we reach an initial state during this analysis, we know that the target is reachable from the set of initial states.

Algorithm 1. Algorithm for analyzing an attack jungle

FIND-ALL-ATTACKS(J, t)

```
1  visited ← ∅
2  working ← {t}
3  while working ≠ ∅
4        do choose v ∈ working
5           for each w ∈ Pre⟶(v)
6                 do if ∀u ∈ (visited ∪ working) : u ⋠ w
7                       then working ← working ∪ {w}
8           visited ← visited ∪ {v}
9           working ← working − {v}
10 return visited ∩ M(ℂ)
```

The algorithm takes as input an attack jungle J and a target state t, and returns the set of minimal initial states from which t is reachable. Typically t would be of the form $[v]$, where v is a sink (i.e. have no outgoing edges), or a node close to a sink. Returning to our example in Figure 1, a typical target would be ["send horse head"]. Taking t to include more than one element allows for analysis of simultaneous goals.

Termination. Since the relation $\leq$ on multisets is a *well quasi-order*, our algorithm is guaranteed to terminate [2]. Note that the termination is independent of the structure of the attack jungle. In particular, the existence of cycles in the attack jungle is permitted.

4 Analyzing an Attack Jungle

When we have computed all possible states from which the target is reachable, we can use the framework of [11] to compute certain attributes of the system. The general idea is that we can assign attributes such as cost or difficulty to the attack components, and then automatically compute the cost or difficulty of the attack goal. This is formalized through the use of an *attribute domain*. Intuitively, an attribute domain describes the way in which values are assigned to states, and how to determine which among a set of such values that should be assigned to the final attack goal.

To faithfully model certain attacks, we need to extend the framework of [11]. The reason is that we believe that the concept of *reusability* is crucial to faithfully model certain attacks. Reusability is a property of an attack component, with the meaning that once used, it can be used multiple times at no extra cost. For example, we can think of some information that once acquired, can be taken into account multiple times without being destroyed. This is the case in the attack jungle in Figure 1, where the two attack components "shadow victim" and "dumpster diving" are annotated with a capital **R**, signifying that they are reusable. In

particular, the reusability of "shadow victim" means that we can accomplish both "get address" and "get name" by shadowing the victim only once. Not having reusability can lead to problems in the analysis of the result of an attack jungle, since it can cause the analysis to overestimate the cost of attacks.

We now give some preliminaries of how to compute attributes, closely following the presentation of [11]. An attribute $\alpha : \mathbb{C} \rightarrow \mathbb{D}$ is a function which given a set $\mathbb{D}$ of attribute values, assigns a value to each attack component. To assign a value not only to individual attack components, but also to states, two functions called the *disjunctive* and *conjunctive combinator* are used. The idea is that for each state, we use the conjunctive combinator to combine the attribute values of the attack components. This way, we get an attribute value for each state. We then use the disjunctive combinator to combine the attribute values of the states into one attribute value for the whole attack jungle. For example, if the question we seek to answer is "What is the minimum cost of performing an attack?", a reasonable choice for attribute domain would be $\mathbb{N}$. The attribute α would assign to each attack component a value reflecting the cost of accomplishing that attack component. We would use summation to combine the values for a state, as we need to accomplish all components, paying for each attack component individually. Finally, since we ask for the minimal cost, we would the minimum function to combine the values of all attacks into one value.

We now state this formally. The following definition is taken from [11]:
Let $\mathbb{C}$ be a set of attack components. An attribute domain is a structure $(\mathbb{D}, \bigtriangledown, \bigtriangleup)$ where $\mathbb{D}$ is the set of attribute values, $\bigtriangledown : \mathbb{D} \times \mathbb{D} \rightarrow \mathbb{D}$ is the disjunctive combinator for attribute values and $\bigtriangleup : \mathbb{D} \times \mathbb{D} \rightarrow \mathbb{D}$ is the conjunctive combinator for attribute values. We require that the combinators are associative and commutative, and that $\bigtriangleup$ distributes over $\bigtriangledown$. The associativity and commutativity of the combinators make natural forward to generalize them to multisets.

To handle the concept of reusability, we introduce the *reducing function*, $\rho : \mathcal{M}(\mathbb{C}) \rightarrow \mathcal{M}(\mathbb{C})$. Intuitively, $\rho(m)$ is a multiset identical to m except that for all reusable attack components, all duplicates have been discarded. More formally, for each $r \in m$ such that r is reusable, $\rho(m)(r) = 1$ if $m(r) > 0$ and $\rho(m)(r) = 0$ if $m(r) = 0$. Note that we allow for ρ to depend on the particular attribute that we are considering. This allows an attack component to be reusable with respect to one attribute, but some other another attribute.

We can now define the answer to a question about the attack jungle in the following way. Assume an attribute domain $(\mathbb{D}, \bigtriangledown, \bigtriangleup)$, an attack jungle J and a

Table 1. Examples of attribute domains and corresponding questions

Attribute Domain	Question
$(\{F, T\}, \vee, \wedge)$	Is any attack possible?
$(\{F, T\}, \wedge, \vee)$	Is [some property] needed for all attacks?
$(\mathbb{N}, \min, \max)$	What is the minimum required skill level for an attack?
$(\mathbb{N}, \min, +)$	What is the minimum cost of performing an attack?

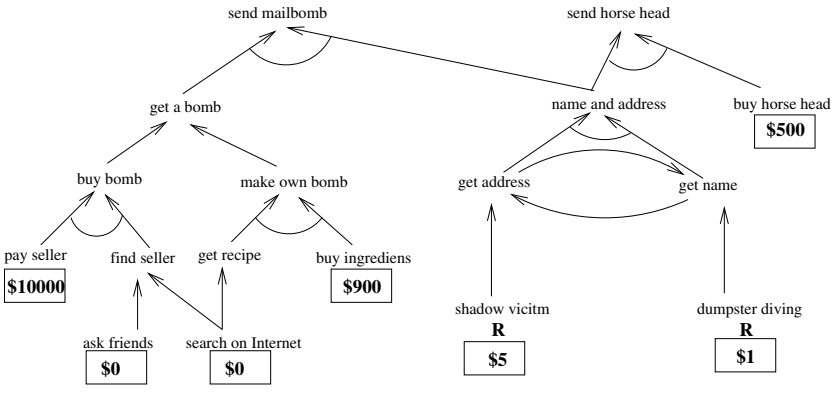


Fig. 2. An attack jungle with attributes

target state t. Let Ω be the set of all minimal states from which we can reach the state t, i.e. the output of our algorithm. Also assume a reducing function ρ. Under these conditions, the answer to the question represented by the attribute domain is

$$\bigtriangledown_{s\in\Omega} \bigtriangleup_{c\in\rho(s)} \alpha(c)$$

Table 1 shows some examples of attribute domains and corresponding questions.

We conclude this section by applying our analysis to the attack jungle in Figure 2.

In Table 2, we give possible attributes to the attack components of the attack jungle. We also provide labels for the attack components to enable compact representation of states. The question we are trying to answer in this example is "What is the minimum cost of performing an attack?", and thus we have $(\mathbb{N}, \min, +)$ as our attribute domain. In Table 3, we give the result of the analysis for two different target states.

This example illustrates why reusability can be needed to model attacks. As can be seen from Table 3, the resulting number would be too large if we simply applied our combinator functions to the minimal initial states. This is

Table 2. Labels and costs for the attack components

Name of attack component	label	cost (α)
Pay seller	a	\$10000
Ask friends	b	\$0
Search on Internet	c	\$0
Buy ingredients	d	\$900
Shadow victim	e	\$5
Dumpster diving	f	\$1
Buy horses head	g	\$500

Table 3. Initial states and costs for the possible threats

Name of attack goal	possible initial states	reduced initial states	cost
Send mailbomb	[a, b, c, d, e, e]	[a, b, c, d, e]	$10905
	[a, b, c, d, e, f]	[a, b, c, d, e, f]	$10906
	[a, b, c, d, f, f]	[a, b, c, d, f]	$10901
	[a, c, c, d, e, e]	[a, c, c, d, e]	$10905
	[a, c, c, d, e, f]	[a, c, c, d, e, f]	$10906
	[a, c, c, d, f, f]	[a ,c, c, d, f]	$10901
		Minimum cost:	$10901
Send horse's head	[e, e, g]	[e, g]	$505
	[e, f, g]	[e, f, g]	$506
	[f, f, g]	[f, g]	$501
		Minimum cost:	$501

very problematic since we are trying to find the *minimum* cost, and therefore have to be very conservative in our estimations.

5 Case Study: The GSM Network

This section gives a brief description on how attack jungles were used for analyzing the security of the GSM network with focus on the radio access network, called GERAN.

First we give a short description of the GSM system. For a more detailed description of the GSM system see [16,8].

5.1 The GSM System

Global System for Mobile communications (GSM) is the most common standard for mobile phones in the world. This study focus on the parts of the GSM system that provides basic connectivity and hence leaves parts of the network related to higher layer services out. The GSM system can be grouped into three main parts: the user equipment, the radio access network, and the core network.

User equipment. The user equipment (UE) is the part of the GSM network that a user operates. It consists of a smart card used in the subscriber authentication process and the device used to access the network. The device could be a mobile phone or some other equipment with at GSM modem.

The Radio Access Network (GERAN). The radio access network is the part of the GSM network that provides radio connectivity for the UE to the GSM network. It consists of two elements, the Base Transceiver Station (BTS) and the Base Station Controller (BSC). The BTS consists of the radio transmitter receivers, and their associated antennas that communicate directly with the user equipment and is the node that is sometimes referred to as a base station. The BTS provides encryption of the traffic to and from the UE. The BSC controls a group of BTSs. It manages the radio resources and controls items such as allocation of channels and handover within the group of BTSs.

The core network. The core network contains a variety of different elements. It provides the main control and interfacing for the whole mobile network. There are two types of accesses, one circuit switched and one packet switched. On a very high level, the difference between circuit switched access and packet switched access is that in the case of circuit switched, a channel is allocated between the two communicating parties when the connection is set-up, and the capacity of the channel cannot be used for other purposes as long as the connection is up. In the case of packet switched, the data transmitted between two communicating parties is packetized and the connection only uses as much capacity as there is data to be transmitted. This gives a more effective utilization of the network resources. The packet switched service was added to the GSM system at a later stage, and introduced two new nodes in the architecture.

The main element of the circuit switched part of the core network is the Mobile switching Services Center (MSC). The MSC provides functionality to enable the requirements of a mobile user to be supported. These include registration, authentication, call location, and call routing to a mobile subscriber. To enable the MSC to perform its functions it requires data from a number of databases. One is the Visited Location Registry (VLR) that contains information about the identities of the UEs that are currently in the area that is served by this network. To make access faster and more convenient, the VLR is (commonly) integrated in the MSC. Another database is the Home Location Register (HLR). It contains administrative information about each subscriber together with their last known location.

The packet switched part of the core network mainly adds two new nodes compared to the circuit switched part; it contains one node called the Serving GPRS Support Node (SGSN). If the UE is attached to the packet switched domain, it is the SGSN that handles the authentication of the user equipment

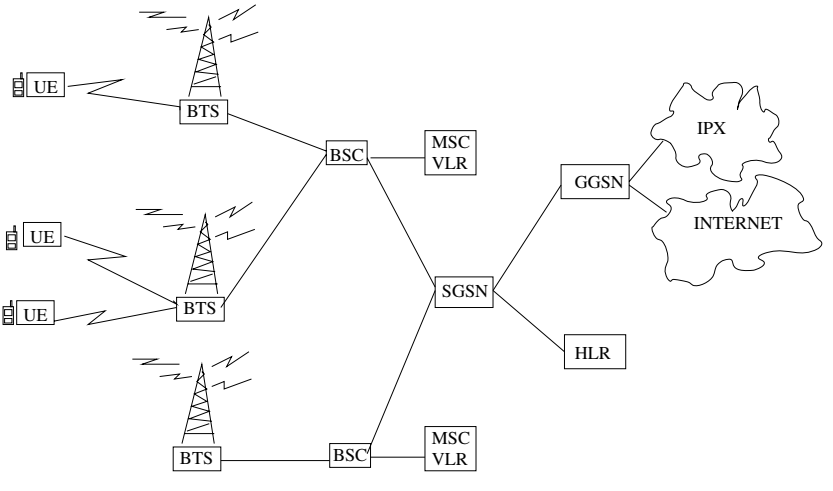


Fig. 3. The architecture of the GSM system

by using information from the HLR database. It is also the SGSN that performs encryption of the data to and from the UE in the packet switched domain. Typically the SGSN is located in a trusted, physically protected place. This gives a higher security compared to the circuit switched case where the encryption is terminated in the BTS, which can be located in physically insecure places (and the links from the BTS to the core network do not have a standardized protection mechanism; sometimes micro wave links are also used for this).

The Gateway GPRS Support Node (GGSN) is the node that provides the UE with an IP point of presence when it is attached to the packet switched domain, it interfaces to external IP networks. The GGSN also collects billing information, provides packet filtering and is responsible for GPRS session management. The architecture of the radio part of the GSM network is shown in Figure 3.

The GSM network provides two main security services to the user: subscriber authentication and encryption of the data sent to and from the UE. In addition to this, there is identity protection by the use of temporary identities.

5.2 Creating the GSM Attack Jungle

Security experts studied the GSM network (focusing on GERAN) carefully and possible threats and corresponding attacks towards the security of the system were realized.

All attacks were modeled in an attack jungle using our tool. The complete attack jungle contained 323 nodes.

Table 4 provides a short description of some of the threats (and corresponding attacks) that were identified by the security experts as possible to realize. For a more detailed description containing all the identified threats (and attacks) that were identified in this case study see [1]. The threat "false base station", was already known to be a quite serious threat to GSM, which in fact lead to countermeasures taken in 3G networks (UMTS). Nevertheless, re-confirmation of known security issues is also useful as it helps to validate the model and analysis.

5.3 Analyzing the GSM Attack Jungle

In this particular study, each attack component in the created attack jungle was ranked with the probability that it might occur and each threat was ranked with the seriousness (how big the damage would be if the threat was realized).

Ranking the probability. Each attack component was ranked on a scale from one to five, where rank one means "negligible" and rank five "almost certain". An example of an attack with rank one is an attack that is successful with a probability of guessing or breaking an 80bit (cryptographic) key. An attack of rank 5 is an attack that can be performed by single, averagely skilled "hackers" with standard PC/phone resources, possibly using "attacking tools" developed by someone else found on the Internet.

Table 4. A small set of the identified threats and their corresponding attacks

Threat	**Attack(s)**
Sensitive information regarding the user is revealed.	- The UE is fooled to reuse a previously compromised cryptographic key. - The UE is/will be fooled to reuse the same cryptographic key with an insecure encryption algorithm. - The cryptographic key is disclosed by other means.
False base station (BTS)	- Attacker disables (for example cutting antenna connection or cutting the power) a real base station and puts up a false base station. Attacker can now fake a base station towards the network and fake a network towards the UE. The attacker can then easily get the information that is sent.
The UEs are not able to use signalling towards the network.	- An attacker sets up a false BSC (could be implemented in a false base station), that broadcasts a "barred access class" message (unencrypted). This messige disables signalling between the network and a set of UEs. UEs does not try to reconnect after receiving theis message (except for emergency calls).
UE is forced to use a different network.	- An attacker changes an "location update accept" message to an "location update reject" with a cause code of "PLMN not available". This forces the UE to look for another network (this is a man-in-the-middle attack).
The UE accepts faked authentication signalling messages.	- The attacker sends a random number (RAND) that is already used (replayed) to the UE.
Successful impersonation of a subscriber	- For an attacker to sucessfully impersonate a subscriber, cryptanalysis of one of the encryption algorithms that are used has to be done.
Long-term subscriber key is disclosed	- The key is disclosed by preforming cryptanalysis of a encryption algorithm (AKA). - The key is leaked from manufacturer. - The key is leaked when it is installed in the autentication center.

Ranking the Seriousness. The seriousness of the threats (or attack goals) were estimated and ranked on a scale from one to five where one means "minimal" and five "very high". Attack goals of rank one are threats that would not imply anything for user privacy, QoS (Quality of Service), or charging, e.g. being able to occasionally increase a phones transmit power. Attack goals of rank five are attacks that if they are realized it would make frontpage news and seriously damage the trust in mobile networks, either from users or operators point of view.

Risk Assessment. By computing the product of the seriousness and the probability for each threat a measure for the risk can be deduced.

For threats with the highest risk protection mechanism (countermeasures) to counter the identified risks can be designed.

Countermeasures. Analyzing the attack jungle and determining the high risk threats gives the opportunity to design and evaluate possible countermeasures that are suitable for the system.

In this case study, some of the possible identified countermeasures for the high risk threats are:

- Add new encryption algorithms.
- Remove insecure encryption algorithms.
- Add integrity protection to broadcasted messages.
- Enlarge the size of encryption keys.
- Add integrity protection to broadcast signalling.

A more detailed description of all the identified countermeasures can be found in [1].

To analyze the effect of the identified countermeasures, each countermeasure was added to the attack jungle. The attack jungle was analyzed using our backward reachability algorithm and the security experts could assure that the countermeasures worked as expected. Each countermeasure was evaluated separately and possible advantages, disadvantages, main vulnerabilities and remaining security issues carefully analyzed.

6 Conclusion

We have introduced a formal, methodical way of describing the security of systems called attack jungles. In this model we allow multiple roots, cycles and reusability of resources. The analysis of an attack jungle is done in two steps; first we extract all possible paths in the attack jungle that leads to an attack goal. Secondly, we assign attributes to the attack components in the extracted paths. By decorating the attack components with different attributes we can answer questions such as "What is the minimum cost of performing an attack?" and "What attack is most likely to happen?"

Previously, analyzing the security of systems was traditionally done by hand. However when analyzing a large and complex system such as the GSM network this approach is impossible and the notion and implementation of attack jungles was a necessary and helpful tool for analyzing the security of such a system. Using the ability to model the security of system with cycles, multiple roots and reusable nodes both modeling and the analysis was significantly simplified. The backward reachability algorithm described in this paper easily verifies properties of the attacks. Countermeasures can be added to the model and the algorithm can verify that each countermeasure works as expected and also analyze the behavior of different combinations of several countermeasures.

We have tested our approach by implementing a prototype tool where we can both model and analyze attack jungles. The tool have been used to model possible attacks towards the security in the GSM radio network. A set of possible threats and corresponding attacks towards the security in the GSM radio network where modeled using the tool. For the attacks with the highest risk,

countermeasures were designed and the tool was used to evaluate the impact/benefit of implementing the different countermeasures.

When analyzing the attack jungle and the designed countermeasures, the analysis showed that some of the countermeasures was not secure enough to remove all attacks. This should be considered when updating the system since the money and effort put into implementing the countermeasure might not be reasonable.

Several extension can be carried out to the attack jungle formalism. We plan to extend the model by adding weights to each edge in the tree. The weight could for example symbolize how important an attack component are for the ability to accomplish a specific attack. Another direction for future work is to encode attack jungles symbolically and extend our algorithm to work on a symbolic encoding of the attack jungle.

References

1. Specification of GERAN vulnerabilities (2007), http://www.3gpp.org/ftp/Specs/html-info/33801.htm
2. Abdulla, P.A., Čerāns, K., Jonsson, B., Tsay, Y.: Algorithmic analysis of programs with well quasi-ordered domains. Inf. Comput. 160(1-2) (2000)
3. Isograph AttackTree+ (2007), http://www.isograph-software.com/atpover.htm
4. Bistarelli, S., Dall'Aglio, M., Peretti, P.: Strategic games on defense trees. In: Dimitrakos, T., Martinelli, F., Ryan, P.Y.A., Schneider, S. (eds.) FAST 2006. LNCS, vol. 4691, pp. 1–15. Springer, Heidelberg (2007)
5. Bistarelli, S., Fioravanti, F., Peretti, P.: Defense tree for economic evaluations of security investment. In: ARES 2006, pp. 416–423. IEEE Computer Society, Los Alamitos (2006)
6. Bistarelli, S., Peretti, P., Trubitsyna, I.: Analyzing security scenarios using defence trees and answer set programming. Electronic Notes in Theoretical Computer Science (ENTCS) 197(2), 121–129 (2008)
7. Brynielsson, J., Horndahl, A., Kaati, L., Mårtenson, C., Svenson, P.: Development of computerized support tools for intelligence work. In: Proceedings of ICCRTS (2009)
8. Goleniewski, L., Jarrett, K.W.: Telecommunications Essentials, Second Edition: The Complete Global Source, 2nd edn. Addison-Wesley Professional, Reading (2006)
9. Van Lamsweerde, A., Brohez, S., De Landtsheer, R., Janssens, D.: From system goals to intruder anti-goals: Attack generation and resolution for security requirements engineering. In: Proc. of RHAS 2003, pp. 49–56 (2003)
10. Amenaza Technologies Limited. A quick tour of attack tree based risk analysis using SecureITree (2002)
11. Mauw, S., Oostdijk, M.: Foundations of attack trees. In: Won, D.H., Kim, S. (eds.) ICISC 2005. LNCS, vol. 3935, pp. 186–198. Springer, Heidelberg (2005)
12. Lippmann, R., Sheyner, O., Haines, J., Jha, S., Wing, J.M.: Automated generation and analysis of attack graphs. In: Proceedings of the IEEE Symposium on Security and Privacy (2002)

13. Schneier, B.: Attack trees. Dr Dobbs Journal 24(12) (1999)
14. Sheyner, O., Wing, J.M.: Tools for generating and analyzing attack graphs. In: de Boer, F.S., Bonsangue, M.M., Graf, S., de Roever, W.-P. (eds.) FMCO 2003. Sheyner, O., Wing, J.M, vol. 3188, pp. 344–371. Springer, Heidelberg (2004)
15. van Lamsweerde, A.: Elaborating security requirements by construction of intentional anti-models. In: Proc. of ICSE 2004, pp. 148–157 (2004)
16. Wisniewski, S.: Wireless and Cellular Networks. Prentice-Hall, Inc., Englewood Cliffs (2004)

Formal Modeling and Verification of Sensor Network Encryption Protocol in the OTS/CafeOBJ Method

Iakovos Ouranos[1,2], Petros Stefaneas[3], and Kazuhiro Ogata[4]

[1] School of Elec. and Comp. Eng., National Tech. Univ. of Athens (NTUA)
[2] Hellenic Civil Aviation Authority, Heraklion Airport
[3] School of Appl. Math. and Phys. Sci., National Tech. Univ. of Athens (NTUA)
[4] School of Info. Sci., Japan Adv. Inst. of Sci. and Tech. (JAIST)
iouranos@central.ntua.gr, petros@math.ntua.gr, ogata@jaist.ac.jp

Abstract. Sensor Network Encryption Protocol (SNEP) is one of the secure building blocks of the SPINS Protocol Suite and provides data confidentiality, two-party data authentication and evidence of data freshness in a wireless sensor network. We have formally analyzed SNEP and a node-to-node key agreement protocol based on it, using the OTS/CafeOBJ method. Among other invariants, we have verified that the protocols possess the important security properties of authenticity and confidentiality of relevant message components. To our knowledge, we are the first to formally analyze SNEP using algebraic specification techniques.

Keywords: Algebraic Specification, Wireless Sensor Networks, Security, CafeOBJ, Observational Transition Systems, Formal Verification.

1 Introduction

Security for wireless sensor networks is a challenging task, since sensors, compared to conventional desktop computers, have limited resources and cannot use the computationally expensive asymmetric cryptography. As a result, the security protocols designed for such settings should provide high level of assurance with the use of symmetric cryptographic primitives. This makes formal analysis of these protocols even more necessary.

Sensor Network Encryption Protocol (SNEP) is the building block of SPINS protocol suite [1] that provides data confidentiality, authentication, integrity and weak message freshness using symmetric cryptographic primitives. In this paper, which is a revised version of [2], we formally analyze SNEP and an application of it to the node-to-node key agreement, using the OTS/CafeOBJ method [3]. The main differences between the work reported in [2] and this paper are:

a) the formal models of the two protocols are different since we have added a data part at each message constructor, the constructors for most data types used are different, and finally, transitions describing the OTS are more.

b) the verified properties are different and more critical for the protocols, since we have verified additionally a secrecy property, while also authentication property is expressed correctly for a more general case.

T. Margaria and B. Steffen (Eds.): ISoLA 2010, Part I, LNCS 6415, pp. 75–89, 2010.

c) we report on related work on formal analysis of sensor network protocols and systems and present some lessons learned.

In the method we use, a protocol, algorithm, or software system is modeled as an Observational Transition System (OTS), which is a kind of transition system that can be written straightforwardly in terms of equations. Next, the OTS is described in CafeOBJ algebraic specification language [4]. Properties to verify are then expressed as CafeOBJ terms, and proof scores showing that the specified OTS model has desired properties are also written in CafeOBJ. Finally, proof scores are executed with the CafeOBJ system. Even if the protocol we analyze here is used in wireless sensor networks, location information does not affect the properties we verify. As a result, we need not model mobility in the specification. But in case mobility of sensors should be taken into account, OTSs can be also used as mathematical models by simply adding the corresponding operators. To our knowledge, we are the first to analyze SNEP using algebraic specification techniques. This work is part of our research on modeling, specifying and verifying mobile systems using formal methods [12].

The rest of the paper is organized as follows: Section 2 explains the OTS/CafeOBJ method, while section 3 describes the protocols to be analyzed. Section 4 and Section 5 deal with the formal modeling and verification of SNEP and Node-to-Node Key Agreement protocol, respectively. Section 6 discusses some lessons learned while Section 7 presents related work. Finally, Section 8 concludes the paper.

2 The OTS/CafeOBJ Method

2.1 Introduction to CafeOBJ

CafeOBJ [4] is an executable algebraic specification language/system. The main underlying logics of CafeOBJ are order-sorted algebras [5] and hidden algebras [6-7]. The former is used to specify abstract data types while the latter to specify abstract state machines, providing support for object oriented specifications. There are two kinds of sorts in CafeOBJ, visible sorts representing abstract data types, and hidden sorts representing the set of states of a state machine. The operations to hidden sorts are classified into actions, observations and hidden constants. An action can change the state of an abstract machine. It takes a state of the abstract machine and zero or more data, and returns another or the same state of the abstract machine. An observation is used to observe the value of a data component in an abstract machine. It takes a state of an object and zero or more data, and returns the value of a data component in the abstract machine. Hidden constants denote initial states of abstract machines. Declarations of observation and action operators start with keyword *bop* or *bops*, and those of other operators with *op* or *ops*. Equations start with *eq* while *ceq* is used for the case of conditional ones. The CafeOBJ system rewrites a given term by regarding equations as left-to-right rewrite rules.

2.2 Observational Transition Systems

An Observational Transition System, or OTS [3], is a kind of transition system that can be written in terms of equations in an algebraic specification language. We assume that there exists a universal state space called Y and we also suppose that each

data type we need to use in the OTS, including the equivalence relationship for that data type, has been declared in advance. An OTS S is the triplet $\langle O, I, T \rangle$ where:

- O: A finite set of observers. Each $o \in O$ is a function $o : Y \rightarrow D$, where D is a data type that may differ from observer to observer. Given an OTS S and two states $u_1, u_2 \in Y$, the equivalence ($u_1 =_S u_2$) between them w.r.t. S is defined as $\forall o \in O. o(u_1) = o(u_2)$.
- I : The set of initial states such that $I \subseteq Y$.
- T : A set of conditional transitions. Each $\tau \in \mathrm{T}$ is a function $\tau : Y \rightarrow Y$, such that $\tau(u_1) =_S \tau(u_2)$ for each $u_1, u_2 \in Y / =_S$. For each $u \in Y$, $\tau(u)$ is called the successor state of u w.r.t. τ. The condition c_τ of τ is called the effective condition. Also for each $u \in Y$, $\tau(u) =_S u$ if $\neg c_\tau(u)$.

Observers and transitions may be parameterized. Generally, observers and transitions are denoted as $o_{i_1,\ldots,i_m}$ and $\tau_{i_1,\ldots,i_n}$ respectively, provided that $m, n \geq 0$ and there exist data types D_k such that $k \in D_k (k = i_1,\ldots,i_m, j_1,\ldots, j_n)$.

An OTS is described in CafeOBJ. Observers are denoted by CafeOBJ observation operators, and transitions by CafeOBJ action operators.

An execution of *S* is an infinite sequence $u_0, u_1, \ldots$ of states satisfying:

1. Initiation: $u_0 \in I$.
2. Consecution: For each $i \in \{0,1,\ldots\}$, $u_{i+1} =_S \tau(u_i)$ for some $\tau \in \mathrm{T}$.

A state is called reachable w.r.t. S if and only if it appears in an execution of S. Let R_S be the set of all the reachable states w.r.t. S. All properties considered in this paper are invariants, which are defined as follows: Any state predicate $p : Y \rightarrow Bool$ is called invariant w.r.t. S if p holds (i.e. is true) in all reachable states w.r.t. S. *Bool* denotes the sort of Boolean values, and is defined in the built-in CafeOBJ module BOOL.

3 The SPINS Protocol Suite

SPINS [1] protocols were designed to support the security requirements of sensor networks. The limited computation resources of sensors make it impossible to use asymmetric cryptography. As a result, the designers of SPINS protocols use purely symmetric cryptographic primitives. The security requirements of sensor networks include data confidentiality, data authentication, data integrity and data freshness. To achieve these security requirements, two protocols were designed and implemented: SNEP and μTESLA. SNEP provides data confidentiality, two-party data authentication, integrity and freshness. μTESLA provides authentication for data broadcast. In addition, they built an authenticated routing application using the μTESLA, and a two-party key agreement protocol, based on SNEP. In this paper we are going to analyze SNEP and the two-party key agreement protocol.

The notation we follow is the same as in [1]:

- A, B are communicating nodes.
- S is a base station.
- N_i, $i \in \{A, B\}$, is a nonce generated by node i. A nonce is an unpredictable bit string, usually used to achieve freshness.
- $M_1 \mid M_2$ denotes the concatenation of messages M_1 and M_2.
- K_{ij}, $i, j \in \{S, A, B\}$, $i \neq j$, denotes the secret encryption keys shared between i, j. The way keys are produced is described in [1].
- R_i, $i \in \{A, B\}$, denotes some data produced by node i.
- C_i, $i \in \{A, B\}$, denotes a counter of a message sent from i.
- $\{M\}_{K_{ij}}$, $i, j \in \{S, A, B\}$, $i \neq j$, denotes the encryption of message M with symmetric key shared by nodes i and j.
- $\{M\}_{<K_{ij}, IV>}$, $i, j \in \{A, B\}$, $i \neq j$, denotes the encryption of message M with the key K_{ij} and the initialization vector IV which is used in encryption modes such as cipher-block chaining (CBC), output feedback mode (OFB), or counter mode (CTR).
- $MAC(K_{ij},M)$, $i, j \in \{S, A, B\}$, $i \neq j$, denotes the message authentication code of message M, encrypted with key K_{ij}.

3.1 The Sensor Network Encryption Protocol

The entire SNEP protocol, works as follows: If a node A wants to authenticate node B, it generates randomly a nonce N_A and sends it along with R_A to B. On receipt of the message, B obtains N_A and R_A and sends back R_B encrypted with shared key K_{BA} and the counter C_B, which is the initialization vector, has different value for each message and direction of communication and is shared between the two parties. With the use of counter, the protocol achieves the so-called *semantic security* which ensures that an intruder has no information about the plaintext, even if it sees multiple encryptions of the same plaintext. The messages exchanged are:

$$A \rightarrow B : N_A, R_A,$$

$$B \rightarrow A : \{R_B\}_{\langle K_{BA}, C_B \rangle}, MAC(K'_{BA}, N_A \mid C_B \mid \{R_B\}_{\langle K_{BA}, C_B \rangle}).$$

Additionally, to optimize the process of achieving strong data freshness, instead of simply returning the nonce with the response message R_B in an authenticated protocol, B sends N_A, C_B and the encrypted R_B in a message authentication code (MAC) encrypted with K'_{BA}. On receipt of the message, A decrypts it, obtains R_B and verifies the MAC. If the verification succeeds, then it is sure that B generated the response after it sent the request. If confidentiality and data authentication are needed also for the first message, then the message can be altered as follows:

$$A \rightarrow B : \{N_A \mid R_A\}_{\langle K_{AB}, C_A \rangle}, MAC(K'_{AB}, C_A \mid \{N_A \mid R_A\}_{\langle K_{AB}, C_A \rangle})$$

Apart from *semantic security*, SNEP provides:

- *Replay protection*: The counter value in the MAC prevents replaying old messages. If the counter were not present in the MAC, an adversary could easily replay messages.
- *Strong freshness*: If the MAC is verified successfully, node *A* knows that node *B* generated the response after it sent the request.
- *Data Authentication*: To achieve two-party authenticity, the protocol uses a MAC. If the MAC verifies correctly, the receiver can be assured that the message originated from the claimed sender.
- *Low communication overhead*: The counter state is kept at each point and does not need to be sent in each message.

3.2 The Node-to-Node Key Agreement Protocol

To bootstrap secure connections, a protocol for symmetric key setup is needed. The protocol has been constructed solely from symmetric key algorithms. It uses a base station as a trusted agent for key setup.

Assume that node *A* wants to establish a secret session key SK_{AB} with node *B*. Since *A* and *B* do not share any secrets, they need to use a trusted third party *S*, which is the base station. In the trust setup, both *A* and *B* shared a secret key with the base station K_{AS} and K_{BS}, respectively. The protocol shown below achieves secure key agreement as well as strong key freshness.

$$A \rightarrow B : N_A, A$$
$$B \rightarrow S : N_A, N_B, A, B, MAC(K_{BS}, N_A \mid N_B \mid A \mid B)$$
$$S \rightarrow A : \{SK_{AB}\}_{K_{AS}}, MAC(K_{AS}, N_A \mid B \mid \{SK_{AB}\}_{K_{AS}})$$
$$S \rightarrow B : \{SK_{AB}\}_{K_{BS}}, MAC(K_{BS}, N_B \mid A \mid \{SK_{AB}\}_{K_{BS}})$$

The protocol is an application of SNEP with strong key freshness. Nonces N_A and N_B ensure strong key freshness to both *A* and *B*. The SNEP protocol is responsible to ensure confidentiality of the established session key SK_{AB}, as well as message authenticity to make sure that the key was really generated by the base station. MAC in the second message helps defend the base station from Denial-of-Service attacks (DoS), so the base station only sends two messages to *A* and *B* if it received a legitimate request from one of the nodes.

4 Formal Modeling and Verification of SNEP

4.1 Modeling

We suppose that there exist untrustable nodes as well as trustable ones. Trustable nodes exactly follow the protocol, but untrustable ones may do something against the protocol as well, namely eavesdropping and/or faking of messages. The combination

and cooperation of untrustable nodes is modelled as the most general intruder [8]. The cryptosystem used is perfect and so, the intruder can do the following:

- Eavesdrop any message flowing in the network.
- Glean any nonce, data, ciphertext and message authentication code (MAC) from the message; however the intruder can decrypt a ciphertext only if he knows the corresponding key to decrypt.
- Fake and send messages based on the gleaned information; however the intruder cannot guess unknown nonces and data.

We first formalize data types that constitute messages in terms of order-sorted algebras. We declare the following visible sorts and the corresponding data constructors for those data types:

- *Node* denotes sensor nodes. Constant *enemy* denotes the intruder.
- *Rand* denotes random numbers which makes nonces unguessable and unique.
- *Nonce* denotes nonces. Given nodes a, b and a random number *r*, *n(a,b,r)* denotes the nonce created by *a* for *b*. Projections *creator*, *forwhom* and *rand* return the first, second and third arguments.
- *Data1, Data2* denote the data R_A and R_B, respectively. Given node *a*, *d(a)* denotes the data created by node *a*. Projection *n* returns the node created the data.
- *Key* denotes symmetric keys shared by two nodes. Given nodes *a*, *b k(a,b)* denotes the key shared by the nodes. Operators *s*, *r* return the first and second arguments.
- *Mackey* models the keys used for the creation of message authentication codes.
- *Counter* denotes the counter shared by two nodes. Given nodes *a, b, c(a,b)* is the data constructor of the counter.
- *Cipher* denotes ciphertexts. Given a symmetric key *k*, a counter *c* and data of the type *Data2 d*, *enc(k,c,d)* denotes the ciphertext obtained by encrypting *d* with *k* and *c*. Operators *k*, *c*, and *d* return the first, second and third arguments.
- *Mac* denotes MACs. Given a *Mackey k*, a *Nonce n*, a *Counter c* and a *Cipher ci*, *mac(k,n,c,ci)* denotes the MAC obtained by encrypting nonce, counter and cipher with *k*. Operators *k, c, n, ci* return the arguments of *mac(k,n,c,ci)*.

In addition to the above visible sorts, we use the visible sort *Bool* that denotes truth values, declared in the built-in module *BOOL*.

Formalization of Messages. The operators to denote the two kinds of messages are declared as follows:

op m1 : Node Node Node Nonce Data1 -> Msg
op m2 : Node Node Node Cipher Mac -> Msg

The visible sort *Msg* denotes messages. Projections *crt*, *src*, *dst* return the first (actual creator), second (seeming sender), and third (receiver) arguments of each message. The first argument is meta-information that is only available to the outside observer and the node that has sent the corresponding message, and cannot be forged by the

intruder, while the remaining arguments may be forged by the intruder. So, if the first argument is the *enemy* and second one is not, then the message has been faked by the intruder. A predicate $mi?$, $i \in \{1,2\}$, checks if the given message is of the type *m1* or *m2*. Projections *n*, *d* return the fourth and fifth argument of the first message, while *c*, *m* the fourth and fifth argument of the second message.

Formalization of the Network. The network is modeled as a multiset of messages, which is used as the storage that the intruder can use. Any message that has been sent or put into the network is supposed to be never deleted from the network. As a consequence, the emptiness of the network means that no messages have been sent.

The intruder tries to glean six kinds of quantities from the network. These are the nonces, the two kinds of data, the ciphers, the message authentication codes and the counters. The collections of these quantities are denoted by the following operators:

```
op nonces : Network -> ColNonces . op ciphers : Network -> ColCiphers .
op data1 : Network -> ColData1. op data2 : Network -> ColData2 .
op macs : Network -> ColMacs . op counts : Network -> ColCounts .
```

Network is the visible sort denoting networks. Col*X* is the visible sort denoting collections of quantities denoting by visible sort *X* (*X* = *Nonce, Cipher, Data1, Data2, Mac, Counter).* For example, given a snapshot *nw* of the network, *nonces(nw)* and *macs(nw)* denote the collection of nonces and message authentication codes available to the intruder.

Those operators are defined with equations. For the case of *nonces* the equations are as follows:

```
eq N \in nonces(void) = (creator(N) = enemy) .
ceq N \in nonces(M,NW) = true  if m1?(M) and n(M) = N .
ceq N \in nonces(M,NW) = true if m2?(M) and
                          (s(k(m(M))) = enemy or r(k(m(M))) = enemy) and
                           n(m(M)) = N .
ceq N \in nonces(M,NW) = N \in nonces(NW)  if not(m1?(M) and n(M) = N) and
                           not(m2?(M) and (s(k(m(M))) =  enemy or r(k(m(M))) = enemy)
                          and n(m(M)) = N) .
```

Constant *void* denotes the empty bag, while *N, M, NW* are CafeOBJ variables for *Nonce*, *Msg* and *Network*, respectively. Operator _\in_ is the membership predicate of collection, while _,_ is the data constructor of bags. So, *M,NW* denotes the network obtained by adding message *M* to the network *NW*. The first equation says that initially, the intruder's nonce is the only available to him. The second equation says that if there exists a message *M* of the type *m1* in the network, then the nonce *N* of the message is available to the intruder, since it is sent in clear. Third equation defines the case in which the intruder can obtain a nonce from message of the second type. The last equation says that in the rest cases, the intruder cannot get any nonces from the message *M*.

The equations defining *macs* are:

```
eq MC \in macs(void) = false .
ceq MC \in macs(M,NW) = true if m2?(M) and
                          not(s(k(m(M))) = enemy and r(k(m(M))) = enemy)
                          and MC = m(M) .
ceq MC \in macs(M,NW) = MC \in macs(NW)  if not(m2?(M) and
                        not(s(k(m(M))) =  enemy and r(k(m(M))) = enemy)
                        and MC = m(M)) .
```

The first equation says that initially no *macs* are available to the intruder. Messages including *macs* are only *m2* messages. If there exists an *m2* message and the *mac* in the message is not encrypted with the intruder's key, then the intruder gleans the *mac*. If the *mac* is encrypted with the intruder's key, the intruder does not have to glean it, because he can reconstruct it from the quantities that constitute it. If a message is not *m2*, any macs cannot be gleaned from the message, and if the *mac* in a *m2* is encrypted with the intruder's key, the intruder does not glean the *mac*, which is denoted by the last equation.

Equations defining the remaining operators are written likewise.

Formalization of Trustable Nodes. The behavior of the protocol is defined in the main module of the OTS specification called *SNEP*. The state space of the protocol is denoted by the hidden sort *Snep*. The values observable from the outside of the protocol are the set of used random numbers and the network. These values are denoted by CafeOBJ observation operators *ur* and *nw*, respectively:

```
bop ur : Snep -> Urand . bop nw : Snep -> Network .
```

where *URand* is the visible sort denoting sets of random numbers. A really fresh random number does not exist in the set of used random numbers.

The behaviour of trustable nodes is sending the two kinds of messages following the protocol and is denoted by two CafeOBJ action operators:

```
bop sdm1 : Snep Node Node Rand  -> Snep . bop sdm2 : Snep Node Msg Rand -> Snep
```

The operators are defined with equations. For example, the equations defining *sdm2* are shown below:

```
op c-sdm2 : Snep Node Msg Rand -> Bool
eq c-sdm2(S, P2, M1,R)
   = (M1 \in nw(S) and m1?(M1) and P2 = dst(M1) and
     n(src(M1),P2,R) = n(M1) and rand(n(M1)) = R) .
ceq nw(sdm2(S, P2, M1,R))
    = (m2(P2, P2, src(M1), enc(k(P2,src(M1)),
          c(P2,src(M1)), d(P2)), mac(kmac(P2,src(M1)),
          n(src(M1),P2,R),  c(P2,src(M1)),
          enc(k(P2,src(M1)),c(P2,src(M1)),d(P2)))),nw(S))  if c-sdm2(S, P2, M1,R) .
eq ur(sdm2(S, P2, M1,R))  = ur(S) .
bceq sdm2(S, P2, M1,R)   = S if not c-sdm2(S,P2,M1,R)  .
```

The condition *c-sdm2(S, P2, M1,R)* means that there exists a valid *m1* message denoted by *M1* in the network, which has been sent to node *P2*, and random number *R* has been used to create *M1*. If *sdm2* is applied when the condition holds, the set of

used random does not change, which is denoted by the third equation, and message *m2(P2,P2,src(M1),…)* is added to the network (see the second conditional equation). The last conditional equation says that nothing changes if sdm2 is applied when the condition does not hold.

Formalization of the Intruder. Part of the intruder has been modeled as the network. The intruder can send faked messages based on the information he gleans. Transition faking messages are denoted by CafeOBJ action operators. We have 4 transitions modeling faking *m1* and one transition modeling faking *m2*. Action operator *fkm21* denotes faking message *m2* and is defined with equations as follows:

```
op c-fkm21 : Snep Node Node Cipher Mac -> Bool
eq c-fkm21(S, P1, P2, CI, M) = (CI \in ciphers(nw(S)) and  M \in macs(nw(S))) .
ceq nw(fkm21(S, P1, P2, CI, M)) =   (m2(enemy, P1, P2, CI,  M),nw(S))
                                    if c-fkm21(S, P1, P2, CI, M) .
eq ur(fkm21(S, P1, P2, CI, M)) = ur(S) .
ceq fkm21(S, P1, P2, CI, M)  = S if not c-fkm21(S,P1,P2,CI,M) .
```

The above action operator is obtained as follows: Since the body of an *m2* message consists of ciphers and macs, all needed is that *Cipher* and *Mac* is available to the intruder, or can be computed by the intruder. Therefore, there are the following possible ways to fake *m2* message:

1. Using a *Mac* and a *Cipher* available to the intruder.
2. Computing a *Mac* and/or *Cipher* based on gleaned quantities.

In the second way, the intruder has to compute macs and/or ciphers based on keys that he owns. But these kinds of messages are refused by the other node, so it is meaningful to add them in the specification. Consequently, the first way is used to model faking *m2* messages.

The remaining operators are defined likewise.

4.2 Verification

We have verified the following properties:

1. If a node receives a valid message *m2* in response to a message sent by him, the message always originates from the claimed sender.
2. The data sent by the node B in the second message (i.e. R_B) cannot be leaked.

Formally, the above properties are defined as invariants as following:

a. For any reachable state *s*, any three nodes *y*, *p1*, *p2*, any random number *r*,

```
not (p2 = enemy) and not (p1 = enemy) and m1(p1,p1,p2,n(p1,p2,r),d(p1)) \in nw(s) and
m2(y,p2,p1,enc(k(p2,p1),c(p2,p1),d(p2)),
    mac(kmac(p2,p1), n(p1,p2,r), c(p2,p1), enc(k(p2,p1), c(p2,p1), d(P2)))) \in nw(S)
 implies
 m2(P2,P2,P1,enc(k(P2,P1),c(P2,P1),d(P2)),
     mac(kmac(P2,P1), n(P1,P2,R), c(P2,P1), enc(k(P2,P1), c(P2,P1), d(P2)))) \in nw(S) .
```

b. For any reachable state *s*, any data2 *d*,
d \in data2 (nw(s)) implies n(d) = enemy .

Property 2 can be stated as that any data2 gleaned by the intruder is the intruder's own. In the definition of Property 1, node *y* (i.e. the actual sender of *m2*) may be different from *p1* (i.e. the seeming sender of *m2*), implying that *y* is the enemy.

We have used five more state invariants as lemmas to prove the above properties and four lemmas on data types.

In the following, we describe an inductive case of the proof of property 1, where it is shown that *fkm21* preserves the property. The inductive case needs another invariant which is called property 102, to strengthen the inductive hypothesis:

For any reachable state *S*, any nodes *p1*, *p2*, any random number *r*,

not (P1 = enemy) and
mac(kmac(P2,P1),n(P1,P2,R),c(P2,P1),enc(k(P2,P1),c(P2,P1),d(P2))) \in macs(nw(S))
and m1(P1,P1,P2,n(P1,P2,R),d(P1)) \in nw(S) implies
m2(P2,P2,P1,enc(k(P2,P1),c(P2,P1),d(P2)),mac(kmac(P2,P1),n(P1,P2,R),c(P2,P1),
enc(k(P2,P1),c(P2,P1), d(P2)))) \in nw(S)

We declare the operators denoting the properties in a module *INV* and define them with equations. Next, we declare the basic formula to prove in each inductive case of the proof of property 1 and the corresponding equations in a module *ISTEP* as follows:

eq istep1 = inv1(s,y,p1,p2,r) implies inv1(s',y,p1,p2,r) .

istep1 is a constant of *Bool*. *s* and *s'* are constants of *Snep*. The former denotes an arbitrary state and the latter its successor.

The case in which the property 102 is needed to strengthen the inductive hypothesis is as follows:

c-fkm21(s,q1,q2,ci,m) = true
∧ ¬ (p1 = enemy)
∧ ¬ (p2 = enemy)
∧ m1(p1,p1,p2,n(p1,p2,r),d(p1)) \in nw(s)
∧ m2(y,p2,p1,enc(k(p2,p1),c(p2,p1),d(p2)),
mac(kmac(p2,p1),n(p1,p2,r),c(p2,p1),
enc(k(p2,p1),c(p2,p1),d(p2)))) = m2(enemy,q1,q2,ci,m)
∧ ¬ m2(p2,p2,p1,enc(k(p2,p1),c(p2,p1),d(p2)),
mac(kmac(p2,p1),n(p1,p2,r),c(p2,p1),
enc(k(p2,p1),c(p2,p1),d(p2)))) \in nw(s)

For more details on the OTS/CafeOBJ method someone can consult [3].

5 Formal Analysis of Node-to-Node Key Agreement Protocol

Following the same methodology, we have modelled the node-to-node key agreement protocol as an OTS, specified it in CafeOBJ and verified two invariant properties. This protocol is more complex than SNEP, since there are 4 messages in the protocol and apart from sensor nodes, the base station as a participant.

In this case, the quantities that the intruder tries to obtain from the network and from the messages are nonces used in the first and second message, the ciphers of the third and fourth message, the encrypted key shared between the two nodes and is part of the third and fourth message, and the message authentication codes used in the second, third and fourth message.

The properties we verified are as follows:

1. If a node receives a valid message *m3* from a base station *b1* in response to a message sent by him to another node, the message always originates from the claimed base station.
2. The key sent by a base station in a *m3* message cannot be leaked.

The second property is a secrecy property and is formalized in CafeOBJ as follows:

For any reachable state *S*, any node key *NK,*

NK \in keys(nw(S)) implies n1(NK) = enemy or n2(NK) = enemy or b(NK) = ibase

The first property is an agreement property and is formalized as follows:

```
not (N1 = enemy) and not (N2 = enemy) and not (B1 = ibase) and
m1(N1, N1, N2, n(N1,N2,R1) ,N1) \in nw(S) and
m2(X,N2,B1,n(N1,N2,R1),n(N2,B1,R2), N1,N2,
    mac1(k(N2,B1),n(N1,N2,R1),n(N2,B1,R2),N1,N2)) \in nw(S) and
m3(Y,B1,N1,enc(k(N1,B1),k(B1,N1,N2)),mac2(k(N1,B1), n(N1,N2,R1), N2,
    enc(k(N1,B1),k(B1,N1,N2)))) \in nw(S) implies
m3(B1,B1,N1,enc(k(N1,B1),k(B1,N1,N2)),mac2(k(N1,B1),
    n(N1,N2,R1),N2,enc(k(N1,B1),k(B1,N1,N2)))) \in nw(S)
```

We need case splitting and five more invariants as lemmas to prove the property. For example, for the case of transition *fkm31* which models the faking of message *m3* based on the gleaned ciphers and message authentication codes, we need the following lemma to discard a subcase:

```
mac2(k(N1,B1),n(N1,N2,R1),N2,enc(k(N1,B1),k(B1,N1,N2))) \in macs2(nw(S))
and not (N1 = enemy) and not (B1 = ibase) implies
m3(B1,B1,N1,enc(k(N1,B1),k(B1,N1,N2)),mac2(k(N1,B1),n(N1,N2,R1),N2,
    enc(k(N1,B1),k(B1,N1,N2)))) \in nw(S)
```

which means that the existence of a message authentication code used in a message *m3*, which is encrypted with the key shared between a trustful node *N1* and a trustful base station *B1*, formed following the protocol, implies that a message *m3* with the appropriate arguments exists in the network and originates from the claimed base station *B1*.

6 Lessons Learned

When specifying formally complex software systems, syntax and/or logical errors are common. Syntax errors can be tackled by the compiler of the language. But logical errors can be detected through formal verification. In the OTS/CafeOBJ method we

apply, if some data constructor, observation or transition used in the specification is not expressed according to the behaviour of the protocol, then some property that should hold and contains this expression cannot be verified. In this case we should go back and check again our specification expressions, which is a painful and time consuming process. Another possible error of an inexperienced specifier is the incorrect formal expression of a property. This could lead to incorrect conclusions regarding correctness of a protocol's specification. For example, in our case study, during writing proof scores we realized that an expression for the first property of SNEP did not hold for our specification. More specifically, the property was expressed in CafeOBJ terms as follows:

For any reachable state *s*, any four nodes *x*, *y*, *p1*, *p2*, any random number *r*,
not (p2 = enemy) and not (p1 = enemy) and m1(x,p1,p2,n(p1,p2,r),d(p1)) \in nw(s) and
m2(y,p2,p1,enc(k(p2,p1),c(p2,p1),d(p2)), mac(kmac(p2,p1), n(p1,p2,r), c(p2,p1),
enc(k(p2,p1), c(p2,p1), d(p2)))) \in nw(S)
implies m2(p2,p2,p1,enc(k(p2,p1),c(p2,p1),d(p2)), mac(kmac(p2,p1), n(p1,p2,r), c(p2,p1),
enc(k(p2,p1), c(p2,p1), d(p2)))) \in nw(s)

The only difference between this expression and the correct one is in the first argument of *m1* message. But this minor difference changes the meaning of the property from:

- If a node receives a valid message *m2* in response to a message (**really**) sent by him, the message always originates from the claimed sender.

to:

- If a node receives a valid message *m2* in response to a message that (**seems** to) have been sent by him, the message always originates from the claimed sender.

When we tried to verify the second property, we used two more lemmas:

For any reachable state *S*, any three nodes *x, p1*, *p2*, any random number *r*,
not (p1 = enemy) and not (p2 = enemy) and m1(x,p1,p2,n(p1,p2,r),d(p1)) \in nw(s)
implies m1(p1,p1,p2,n(p1,p2,r),d(p1)) \in nw(s) .

and, for any reachable state *s*, any two nodes *p1*, *p2*, any random number *r,*
not (p1 = enemy) and not (p2 = enemy) and r \in ur(s)
implies m1(p1,p1,p2,n(p1,p2,r),d(p1)) \in nw(s) .

The first lemma was used to discard the following subcase for the main property:

c-fkm11(s,q1,q2,r1) = true
∧ ¬ (p1 = enemy) ∧ ¬ (p2 = enemy)
∧ (m1(x,p1,p2,n(p1,p2,r),d(p1)) = m1(enemy,q1,q2,n(q1,q2,r1),d(q1)))
∧ ¬ m2(q2,q2,q1,enc(k(q2,q1),
c(q2,q1),d(q2)),mac(kmac(q2,q1),n(q1,q2,r1),
c(q2,q1),enc(k(q2,q1),c(q2,q1),d(q2)))) \in nw(s)
∧ m2(y,q2,q1,enc(k(q2,q1),c(q2,q1),d(q2)),
mac(kmac(q2,q1),n(q1,q2,r1),c(q2,q1),enc(k(q2,q1),
c(q2,q1),d(q2)))) \in nw(s)

while the second lemma was used to discard the following subcase for the first lemma:

c-fkm11(s,q1,q2,r1) = true
∧ ¬ (p1 = enemy) ∧ ¬ (p2=enemy)
∧ (m1(x,p1,p2,n(p1,p2,r),d(p1)) = m1(enemy,q1,q2,n(q1,q2,r1),d(q1)))
∧ ¬ m1(enemy,q1,q2,n(q1,q2,r1),d(q1)))\in nw(s)
∧ ¬ m1(q1,q1,q2,n(q1,q2,r1),d(q1)))\in nw(s)

But while writing proof scores for the second lemma, CafeOBJ returned false for a subcase, that we could not use a lemma to discard it or proceed to further case splitting. So we could not prove the main property. By observing the cases for which we should use lemmas, we found out that the problem was with cases where *(m1(x,p1,p2,n(p1,p2,r),d(p1)) = m1(enemy,q1,q2,n(q1,q2,r1),d(q1)))*, i.e. when *x* is equal to the enemy *agent*, when applying action *fkm11*. The counterexample is as follows:

Let us assume that sensor node *p1* creates a nonce *n(p1,q1,r)* and sends it along with data *d(p1)* to node *q1* in an initial message *m1*. A malicious node called *enemy* steals *n(p1,q1,r)* and *d(p1)*, creates *m1'* which is equal to *m1(enemy,p1,q1,n(p1,q1,r),d(p1)))* and sends it to *q1*. Upon receiving *m1'*, *q1* replies with a well formed message of the type *m2*, thinking that the node sent *m1'* was *p1*.

This counterexample led us to revise the property expression and substituting it with the correct one, which was proved as presented above.

But we realized that the property expression did not hold after having written a lot of code. A solution to this problem has been proposed in [11], where model checking is combined with interactive theorem proving to falsify and verify systems. More specifically, prior to verification, model checking is applied for a bounded state space of the OTS. If model checking succeeds, then verification is applied for all reachable states. Otherwise, the specification and/or property expression is revised, and model checking is reapplied. In this way, verification process can be more efficient. Since CafeOBJ does not provide model checking capabilities, the specification is written or translated into a language such as Maude [19]. Tools for such specification translation have already been developed.

7 Related Work

The only papers found in the literature related to formal analysis of SNEP are [9-10]. In [10], the authors verify the key agreement protocol based on SNEP using the modal logic of Coffey-Saidha-Newe (CSN) [15] which combines the logic of belief and knowledge to evaluate trust and security of communication protocols. In addition, they propose some additional axioms relating to MAC authentication which are used to simplify the verification of protocols that involve MAC authentication. The verification is performed at the logic level, since no formal executable language or tool is used. In [9], the authors use the HLPSL [13] language used by the AVISPA tool [14] to model check SNEP. They claim that since any intruder can create a false request for a node in a message of the type *m1*, then a resource and bandwidth consumption attack is possible. To prevent this kind of attack, they propose adding a MAC computation over the first message. The authors do not present some lessons learned by the use of AVISPA tool to model check SNEP protocol.

Generally, formal verification of security protocols has been studied by many researchers. The induction-based verification method we applied in this paper has been proposed by CafeOBJ community researchers and several case studies have been successfully conducted [3,16,17]. An alternative simulation-based method has been also proposed for OTSs in [22], where authors claim that the choice of the method depends on the OTS and the invariant property to be verified. Finally, researchers from the Maude community [18], which is a sibling language of CafeOBJ, have applied their methodology based on rewriting logic [19], to the analysis of algorithms used in sensor networks settings [20].

8 Conclusions and Future Work

We have modelled Sensor Network Encryption Protocol and a key agreement protocol based on it as an Observational Transition System in CafeOBJ, and verified with proof scores some important properties of them. The case study demonstrates that the OTS/CafeOBJ method can be applied to the formal verification of security protocols for sensor networks. In addition, we have presented some lessons learned and how a combination of model checking and theorem proving that has been proposed by CafeOBJ community researchers can speed up the verification process.

Our research group works on the formal verification of TESLA [21] protocol, which is a more complex security protocol with real time constraints, to demonstrate the application of the Timed OTS/CafeOBJ method, a real time extension of the OTS/CafeOBJ method, to the formal analysis of such protocols.

References

1. Perrig, A., Szewczyk, R., Tygar, J.D., Wen, V., Culler, D.: SPINS: Security Protocols for Sensor Networks. Wireless Networks 8, 521–534 (2002)
2. Ouranos, I., Stefaneas, P.: Verifying Security Protocols for Sensor Networks using Algebraic Specification Techniques. In: Bozapalidis, S., Rahonis, G. (eds.) CAI 2007. LNCS, vol. 4728, pp. 247–259. Springer, Heidelberg (2007)
3. Ogata, K., Futatsugi, K.: Some Tips on Writing Proof Scores in the OTS/CafeOBJ method. In: Futatsugi, K., Jouannaud, J.-P., Meseguer, J. (eds.) Algebra, Meaning, and Computation. LNCS, vol. 4060, pp. 596–615. Springer, Heidelberg (2006)
4. Diaconescu, R., Futatsugi, K.: CafeOBJ Report. World Scientific, Singapore (1998)
5. Goguen, J.A., Meseguer, J.: Order-sorted algebra I: equational deduction for multiple inheritance, overloading, exceptions and partial operations. TCS 105, 217–273 (1992)
6. Goguen, J.A., Malcolm, G.: A Hidden Agenda. TCS 245, 55–101 (2002)
7. Diaconescu, R., Futatsugi, K.: Behavioural coherence in object-oriented algebraic specification. Journal of Universal Computer Science 6, 74–96 (2000)
8. Dolev, D., Yao, A.C.: On the Security of Public Key Protocols. IEEE Trans. on Inf. Theory 29, 198–208 (1983)
9. Tobarra, L., Cazorla, D., Cuartero, F.: Formal Analysis of Sensor Network Encryption Protocol (SNEP). In: Proc. IEEE MASS 2007, Pisa, Italy, pp. 1–6 (2007)
10. Newe, T.: On the Formal Verification of SNEP Key Agreement Protocol for Wireless Sensor Networks. In: Proc. SENSORCOMM 2007, pp. 186–191 (2007)

11. Ogata, K., Nakano, M., Kong, W., Futatsugi, K.: Induction - Guided Falsification. In: Liu, Z., He, J. (eds.) ICFEM 2006. LNCS, vol. 4260, pp. 114–131. Springer, Heidelberg (2006)
12. Ouranos, I., Stefaneas, P., Frangos, P.: An Algebraic Framework for Modeling of Mobile Systems. IEICE Trans. Fund. E90-A (9), 1986–1999 (2007)
13. Chevalier, Y., et al.: A high level protocol specification language for industrial security-sensitive protocols. In: Proc. SAPS, pp. 193–205 (2004)
14. Viganò, L.: Automated Security Protocol Analysis with the AVISPA Tool. In: Proc. MFPS 2005. ENTCS, vol. 155, pp. 61–86 (2005)
15. Newe, T., Coffey, T.: Formal verification logic for hybrid security protocols. Int. Journal of Comput. Syst. Sci. and Eng. 18, 17–25 (2003)
16. Ogata, K., Futatsugi, K.: Flaw and modification of the iKP electronic payment protocols. IPL 86(2), 57–62 (2003)
17. Ogata, K., Futatsugi, K.: Equational approach to formal analysis of TLS. In: Proc. 25th ICDCS, pp. 795–804 (2005)
18. Clavel, M., et al.: Maude: Specification and Programming in Rewriting Logic. TCS 285(2), 187–243 (2002)
19. Marti-Oliet, N., Meseguer, J.: Rewriting Logic: Roadmap and Bibliography. TCS 285(2), 121–154 (2002)
20. Ölveczky, P.C., Thorvaldsen, S.: Formal Modeling, performance estimation, and model checking of wireless sensor network algorithms in Real-Time Maude. TCS 410(2-3), 254–280 (2009)
21. Perrig, A., et al.: Efficient Authentication and Signing of Multicast Streams over Lossy Channels. In: Proc. IEEE Sec. and Privacy Symp., pp. 56–73 (2000)
22. Ogata, K., Futatsugi, K.: Simulation based verification for invariant properties in the OTS/CafeOBJ method. Electr. Notes Theor. Comput. Sci. 201, 127–154 (2008)

Model-Driven Design-Space Exploration for Embedded Systems: The Octopus Toolset*

Twan Basten[1,2], Emiel van Benthum[2], Marc Geilen[2], Martijn Hendriks[3], Fred Houben[3], Georgeta Igna[3], Frans Reckers[1], Sebastian de Smet[4], Lou Somers[2,4], Egbert Teeselink[2], Nikola Trčka[2], Frits Vaandrager[3], Jacques Verriet[1], Marc Voorhoeve[2], and Yang Yang[2]

[1] Embedded Systems Institute,
[2] Eindhoven University of Technology,
[3] Radboud University Nijmegen,
[4] Océ Technologies
a.a.basten@tue.nl

Abstract. The complexity of today's embedded systems and their development trajectories requires a systematic, model-driven design approach, supported by tooling wherever possible. Only then, development trajectories become manageable, with high-quality, cost-effective results.

This paper introduces the Octopus Design-Space Exploration (DSE) toolset that aims to leverage existing modeling, analysis, and DSE tools to support model-driven DSE for embedded systems. The current toolset integrates Uppaal and CPN Tools, and is centered around the DSE Intermediate Representation (DSEIR) that is specifically designed to support DSE. The toolset architecture allows: (*i*) easy reuse of models between different tools, while providing model consistency, and the combined use of these tools in DSE; (*ii*) domain-specific abstractions to support different application domains and easy reuse of tools across domains.

Keywords: Design-space exploration, Modeling, Analysis, Embedded Systems, CPN Tools, Uppaal.

1 Introduction

High-tech systems ranging from smart phones to printers, from cars to radar systems, and from wafer steppers to medical imaging equipment contain an embedded electronic core that typically integrates a heterogeneous mix of hardware and software components. The resulting platform is often distributed, and it typically needs to support a mix of data-intensive computational tasks with event-processing control components. These embedded components more and more often have to operate in a dynamic and interactive environment. Moreover, not only functional correctness is important, but also quantitative properties related

* This work was carried out as part of the Octopus project with Océ Technologies B.V. under the responsibility of the Embedded Systems Institute. This project is partially supported by the Netherlands Ministry of Economic Affairs under the Bsik program.

T. Margaria and B. Steffen (Eds.): ISoLA 2010, Part I, LNCS 6415, pp. 90–105, 2010.

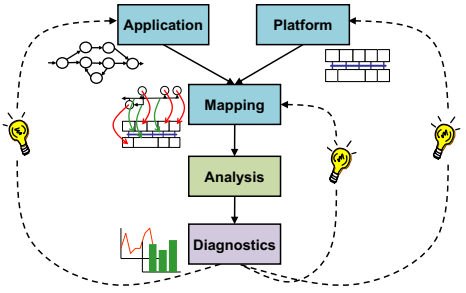


Fig. 1. The Y-chart DSE method

to timeliness, quality-of-service, resource usage, and energy consumption. The complexity of today's embedded systems and their development trajectories is thus increasing rapidly. At the same time, development trajectories are expected to produce high-quality and cost-effective products.

A common challenge in development trajectories is the need to explore extremely large design spaces, involving multiple metrics of interest (timing, resource usage, energy usage, cost). The number of design parameters (number and type of processing cores, sizes and organization of memories, interconnect, scheduling and arbitration policies) is typically very large and the relation between parameter settings and design choices on the one hand and metrics of interest on the other hand is often difficult to determine. Given these observations, embedded-system design trajectories require a systematic approach, that is automated as far as possible. To achieve high-quality results, the design process and tooling needs to be model-driven. No single modeling approach or analysis tool is best fit for all the observed challenges. We propose to leverage the combined modeling and analysis power of various formal methods tools into one integrated Design-Space Exploration (DSE) framework, the *Octopus* framework. The framework is centered around an intermediate representation, *DSEIR* (Design-Space Exploration Intermediate Representation), to capture design alternatives. DSEIR models can be exported to various analysis tools. This facilitates reuse of models across tools and provides model consistency between analyses. The use of an intermediate representation supports domain-specific abstractions and reuse of tools across application domains.

The design of DSEIR follows the Y-chart philosophy [1,16] that is popular in hardware design, see Fig. 1. The Y-chart philosophy is based on the observation that DSE typically involves the co-development of an application, a platform, and the mapping of the application onto the platform. Diagnostic information is used to, automatically or manually, improve application, platform, and/or mapping. DSEIR follows the Y-chart philosophy by supporting, and separating, the specification of its main ingredients, applications, platforms, and mappings. This separation is important to allow independent evaluation of various alternatives of one of these system aspects while fixing the others. Often, for example, various platform and mapping options are investigated for a fixed (set of) application(s).

This paper presents a first version of the Octopus DSE toolset. It integrates CPN Tools [13] for stochastic simulation of timed systems and Uppaal [2] for model checking and schedule optimization. Through an illustrative running example, we present DSEIR, its translations to CPN Tools and Uppaal, and the envisioned use of complementary formal analysis tools in a DSE process. We also evaluated DSEIR on two industrial printer data paths. Simulation and analysis times in CPN Tools and Uppaal for automatically generated models are very similar to the simulation and analysis times for handcrafted models [12].

The paper is organized as follows. We discuss related work in Section 2. Section 3 introduces a typical DSE question, serving as our running example. Section 4 briefly presents the toolset architecture and current realization. DSEIR and the translations to CPN Tools and Uppaal are presented in Sections 5 and 6. Section 7 illustrates the use of the toolset for the running DSE example, and it summarizes the results for the two printer case studies. Section 8 concludes.

2 Related Work

There exists a plethora of academic and commercial frameworks supporting Y-chart-based DSE of embedded systems [4,5,6,7,17,22,23,24,25,28]. Some support formal analysis, in particular the Metropolis/Metro II [5], SHE [25], and Uppaal-based [4,6] frameworks. Others build on simulators, like SystemC or Simulink, and offer no or limited support for other types of analysis such as formal verification or scheduler/controler synthesis. Moreover, most frameworks provide their own modeling and analysis methods and do not support other input and output formats. In contrast, Octopus is open and extensible. Through its intermediate representation DSEIR, it provides a generic interface between input languages and analysis tools; its modeling features can be compared to those of an assembly language (albeit on a much higher abstraction level). Octopus intends to combine well established formal methods in one framework, so that every method can be used for what it is best for. Through its link with Uppaal, it is for example possible to integrate the schedulability analyses of [4,6].

Closest to our work is the Metropolis/Metro II [5] framework. This also aims to be extensible, and to apply formal techniques in DSE. It provides several backends to interface with different analysis tools. The connection with the SPIN model checker in particular can be used to verify declarative modeling constraints expressed in linear temporal logic. The Octopus initiative is complementary to the Metro II framework in the sense that we have integrated different analysis tools. If Metro II is made available to the scientific community, then it should be straightforward to connect the two to combine their strengths.

The already discussed frameworks and tools are all based on the Y-chart approach. There is also a wide range of DSE tools and approaches for specific classes of systems that are not (explicitly) based on the Y-chart. These provide modeling and analysis support and/or automatically explore (parts of) a design space. A good overview of DSE tools can be found in [9]. More recent examples of academic tools are [15,21]. Two industrial frameworks that provide formal modeling and analysis support for DSE are Scade [8] and SymTA/S [10].

Some other DSE frameworks target generic applicability while focusing on generic search and optimization algorithms (such as evolutionary algorithms) needed to explore the design space. Prominent examples are Pisa [3] and Opt4J [19]. These frameworks focus on automating the dashed feedback edges in Fig. 1. The current version of Octopus focuses on the analysis part of the Y-chart, i.e., obtaining the performance metrics for design alternatives. It is complementary to these other frameworks. As future work, we plan to connect Octopus to frameworks like Pisa and Opt4J to automate the exploration of those parts of the design space that are amenable to automatic exploration.

3 Motivating Example

Fig. 2 presents a typical DSE problem. It is used as an illustrative example throughout the paper and as a test case for the Octopus framework. The example shows a pipeline of tasks for processing streams of data objects. The tasks are executed on a heterogeneous multiprocessor platform with two memories and a bus-based interconnect. The example system incorporates ingredients typically observed in today's embedded systems. The task pipeline exhibits dynamic workload variations, depending on the complexity of the data object being processed. Video decoding and encoding, graphics pipelines, and pdf processing for printing are applications showing such data-dependent workload variations. The example platform combines a general-purpose processor (CPU), a graphics processor (GPU) and an FPGA (Field-Programmable Gate Array). Such a mix is often chosen for performance and energy efficiency. An FPGA for example is well suited for executing regular data-intensive computation kernels without much dynamic variation. Multiple kernels can share an FPGA and execute in parallel. A typical DSE question for a system as illustrated in Fig. 2 is to optimize performance while minimizing the resources needed to obtain the performance.

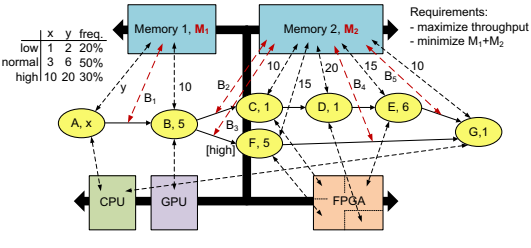


Fig. 2. A running example

The detailed specification of the example system is as follows. The application has tasks A through G, mapped onto a platform consisting of one CPU, one GPU, an FPGA, and two memories, all connected by a bus. Tasks process data objects, one at a time, and need to be executed in an iterative, streaming fashion. The DSE question is to *minimize the memory sizes* M_1 and M_2 such that *throughput* in terms of data objects per time unit is *maximized*, and to find *a simple scheduling policy that achieves this throughput.*

Tasks A and G share the CPU. The order of execution can be chosen freely and preemption is possible. Task B runs on the GPU. Tasks C through F share the FPGA but can be executed in parallel. The annotated dashed arrows between tasks and memories specify memory requirements. The indicated amount is claimed at the task execution start and released at the task completion. The numbers inside tasks indicate the execution time of one execution of the task. A task can only be executed if all required memory is available.

Task A has three execution modes with different workloads (low, normal, high, with average occurrence frequencies of 20%, 50%, and 30%, resp.). The different workloads come with different execution times and memory requirements as indicated in the small table in the figure. Task F is only executed for objects with high workloads, indicated in the figure by the '[high]' precondition.

The edges between tasks indicate data dependencies. Five dependencies involve the transfer of data objects between processing resources. These transfers use the memories, as indicated by the dashed arrows B_1 through B_5. A data object corresponds to 10 memory units. The memory for an object to be produced by a task execution on an output edge is needed at the execution start, to make sure that sufficient space is available. The data object is available for the successor task immediately upon completion of the producing task. The memory of an object being read by a task is released at the execution end, so that the complete data object is available during the entire task execution. The data transfers between tasks C, D, and E are handled inside the FPGA.

For simplicity, we assume that context switching time (on the CPU) and time needed for data transfers over the bus are included in the task execution time.

We would like to leverage existing formal analysis tools to solve the sketched DSE problem. The system combines a stochastic part, the A-B subsystem, and a relatively static part, the B-G subsystem. Simulation tools like CPN Tools are particularly suitable to analyze stochastic (sub)systems, whereas model checking tools like Uppaal are well suited to optimize non-stochastic (sub)systems.

4 The Octopus Architecture and Current Realization

The Octopus toolset aims to be a (*i*) *flexible toolset* that provides (*ii*) *formal analysis support* for DSE by (*iii*) leveraging *existing tools*, also *in combination*; the toolset should be applicable (*iv*) *across application domains* within the embedded-systems domain, and it should allow (*v*) *domain-specific modeling abstractions*. To achieve these goals, the most important architectural choice, illustrated in Fig. 3, is to separate the main ingredients of the Y-chart approach into separate components with well-defined interfaces, centered around an intermediate representation, DSEIR, specifically designed for Y-chart-based DSE. We distinguish three groups of components: (*i*) editing support, for applications, platforms and mapping; (*ii*) analysis support; (*iii*) diagnostics and visualization.

The design of DSEIR is crucial in achieving our goals. It should be sufficiently expressive to allow the specification of design alternatives at the required level of detail. At the same time, it should be well structured and precisely defined, to allow model transformations to various analysis tools that preserve properties of

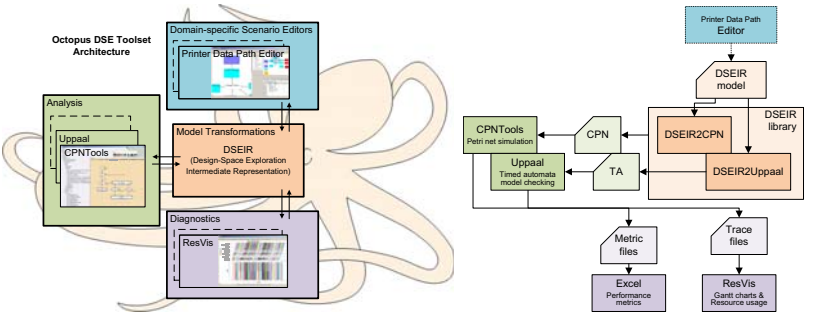


Fig. 3. The Octopus toolset: Architecture (left) and current realization (right)

interest and that provide models that can be efficiently analyzed in the targeted tools. The next two sections explain DSEIR and the transformations to CPN Tools and Uppaal that currently have been implemented. Section 7 provides experimental results that indicate that DSEIR achieves its goals.

The right part of Fig. 3 shows the current Octopus implementation. At the core is a Java library implementing DSEIR. Modules DSEIR2CPN and DSEIR2Uppaal implement the interfaces with CPN Tools and Uppaal. A domain-specific Data Path Editor for modeling printer data paths has been developed and will be integrated in the near future. A printer data path is the electronic heart of a printer that does the image processing for printing, scanning, and copying. Analysis output is visualized through Excel, for (trends in) typical performance metrics such as throughput and memory usage, and ResVis, a simple but powerful tool for visualizing Gantt charts and resource usage over time.

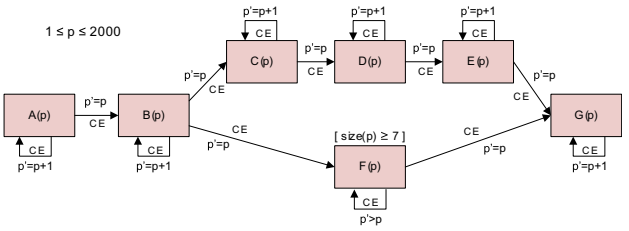
5 DSEIR

DSEIR is implemented as a Java library. Specifications can be entered directly in Java or in an XML format derived from the Java implementation. We present the main principles of DSEIR, focusing on the Java interface. The formal definition of DSEIR and its operational semantics can be found in [26].

Application modeling: DSEIR captures the application side of the Y-chart by means of *Parameterized Timed Partial Orders* (PTPOs). A PTPO is a standard task graph, extended with (*i*) *task parameters* – for simplifying large (and even infinite), albeit structured, precedence relations, (*ii*) *time* – for specifying minimal timing delays between tasks, and (*iii*) four different *types* of precedence rules, for fine-grained specifications in terms of task *enabling* and *completion* events. The expressivity of PTPOs is that of timed event-based partial orders; every PTPO can, in principle, be unfolded to such a, possibly infinite, structure.

Fig. 4a shows the PTPO for our example. Tasks have a parameter p, identifying the objects being processed. Its range is specified in the top left. The condition above F restricts the scope of p for F to objects with high workloads (with size 7–9). A condition $p' = p$ on an edge denotes that the precedence only exists between the same instance of the left and right tasks (the p' is the p of

a)



b)
```
Parameter p = dseir.addParameter("p", 1, 2000);
PTask A = dseir.getApplication().addTask("A", TRUE, p);
  ...
ParameterProperty size_p = p.addProbProperty("Size", 0, 9,
                                    ParameterProperty.Type.PROBABILISTIC);
PTask F = dseir.getApplication().addTask("F", ge(cnst(size_p), cnst(7)), p);
PTask G = dseir.getApplication().addTask("G", TRUE, p);
dseir.getApplication().addEdge(A, A, EdgeType.CE,
                                  eq(cnst(p,true),add(cnst(p),ONE)), ZERO);
dseir.getApplication().addEdge(A, B, EdgeType.CE, eq(cnst(p,true),cnst(p)), ZERO);
```

Fig. 4. a) PTPO for the running example; b) part of its Java DSEIR specification

the right task). The PTPO thus specifies that $A(1)$ should be executed before $B(1)$, $B(1)$ before $C(1)$, etc. There is no direct causal relationship between e.g. $A(2)$ and $C(1)$. The self-loop precedences with condition $p' = p + 1$ eliminate auto-concurrency, i.e., they ensure that no two instances of the same task can be executed at the same time. The self-loop for task F requires condition $p' > p$ because, due to the conditional execution, the range of objects it processes may be non-consecutive. We need only the CE precedence type, specifying a causal relation between the completion (C) of one task and the enabling (E) of another. The other three E-C combinations are typically used for specifying task periods and minimal durations. The example PTPO has no time constraints (all minimal delays zero); we do not impose any delays at the application level, but expect them at the resource level. Fig. 4b shows how the PTPO is specified in DSEIR.

Specifying a platform: A platform in DSEIR is a set of generic *resources*. Each resource has a *name* and a *capacity* (an integer at least zero), and can support preemption or not. No distinction is made between computational, storage and

a)
```
Resource cpu = dseir.getPlatform().addResource("cpu", 1, true);
Resource mem1 = dseir.getPlatform().addResource("mem1", 200, false);
Resource mem2 = dseir.getPlatform().addResource("mem2", 200, false);
```
b)
```
dseir.getMapping().addDuration(A, iF(lt(cnst(size_p),cnst(2)),cnst(1), iF(
                                    lt(cnst(size_p),cnst(7)),cnst(3),cnst(10))));
dseir.getMapping().addUtilizationBound(A, cpu, ONE, ONE);
dseir.getMapping().addUtilizationBound(A, mem1, iF(lt(cnst(size_p),cnst(2)),cnst(12), iF(
       lt(cnst(size_p),cnst(7)),cnst(16),cnst(30))), iF(lt(cnst(size_p),cnst(2)),cnst(12),
                    iF(lt(cnst(size_p),cnst(7)),cnst(16),cnst(30))));
```
c)
```
dseir.getMapping().addHandover(BC, mem2, cnst(10));
dseir.getMapping().addHandover(BF, mem2, iF(lt(cnst(size_p),cnst(7)),cnst(0),cnst(10)));
```
d)
```
dseir.getScheduler().addPriority(A, sub(cnst(1), mult(cnst(10), cnst(p))));
dseir.getScheduler().addPriority(B, sub(cnst(2), mult(cnst(10), cnst(p))));
  ...
dseir.getScheduler().setPreemptive(A, cpu, TRUE);
```

Fig. 5. The running example in DSEIR a) resources; b) duration function; c) handover; d) scheduling

communications resources, nor is any connection between resources explicitly specified. Fig. 5a shows the specification of the CPU and the two memories. Both memories have a capacity of 200 units; the CPU has capacity 1, modeling that it is either free or busy. The preemption flag is true only for the CPU.

Mappings: The mapping part of the Y-chart is captured in DSEIR through the concepts of a *duration function* and a *resource handover function*.

A duration function specifies the duration of a task for any possible resource configuration. If the duration is zero, a task execution is instantaneous once it gets its resources; if the duration is infinite, a task makes no progress with the given resources. The specification of a duration function is split into the specification of the *minimum-required* resource configuration (the unique minimal configuration yielding finite duration), the maximum-required resource configuration (the unique minimal configuration yielding the shortest possible duration) and the actual duration for configurations in between. This can be seen as specifying interval-based resource claims for every task, quantifying resource sharing.

Computational-resource claims that allow sharing are usually specified with the minimum-required configuration of 1 (at least one resource unit is needed), maximum-required configuration of 100 (with less than 100 resource units the task does not run at full pace) and a linear function mapping assigning duration for every capacity in the interval $[1, 100]$. If a task needs exclusive access to a certain amount of resource, as the tasks in our example, then both the minimum-required configuration and the maximum-required one should be equal to the required amount. Storage resources are typically claimed in this way.

Fig. 5b shows the duration function for task A. The first line defines how the duration depends on the parameter of A. The expression has no resource information as A only claims fixed resource amounts. The next lines specify that A needs the CPU exclusively, and that it claims 12, 16 or 30 units of Memory 1 (including the 10 units for output), depending on the current object's size.

Often, a task reads, and subsequently deletes, data that some previous task produced. It is, moreover, sometimes desired to have a task reserve some resource for an upcoming task. DSEIR captures these situations by means of *handover functions*. A handover function for a task specifies the amount of resources that are kept upon its completion for some other task. Handovers are typically assigned to CE edges. Fig. 5c specifies the memory sharing between task B and tasks C and F. The two lines describe that B leaves 10 units of Memory 2 to both C and F, but the latter only when F is to be executed for the same object.

Scheduling: Schedulers are part of the platform in the Y-chart; concrete policies for specific resources and applications are part of the mapping. Nevertheless, DSEIR treats scheduling separately. We predefined a *preemptive*, *event-driven*, *priority-based* scheduler that is activated each time a task is enabled or finishes. The user can specify priorities and the tasks that allow their resources to be taken away at run time (for resources that allow preemption). Fig. 5d shows an example priority assignment for tasks A and B that gives a task processing a lower-numbered object priority over a task processing a higher-numbered one. The code also specifies that task A may be preempted.

Extensions: The current version of DSEIR supports the three main ingredients of the Y-chart approach. Future extensions will provide support for the DSE process itself, allowing the user to specify experiments, properties and performance metrics of interest, and verification and optimization objectives. We also consider developing a language to support the compositional specification of schedulers (e.g., in the style of [18,20]). Complex multiprocessor platforms typically contain multiple interacting schedulers (including arbiters for e.g. buses and memories). Structured language support for scheduling is an enabler for formal analysis of such compositions of schedulers.

6 Model Transformations

The current toolset has interfaces to CPN Tools and Uppaal. This section introduces the model transformations implemented to translate DSEIR models to Coloured Petri Nets (CPNs) and Timed Automata (TA). Based on the semantics of CPN [14], Uppaal TA [27], and DSEIR [26], it can be shown that both translations preserve equivalence; the models generate observation equivalent [11] timed transition systems when considering task enabling and completion events.

6.1 Transforming DSEIR Models to Coloured Petri Nets

CPNs [14] are a well-established graphical formalism, extending classical Petri nets with data, time, and hierarchy. CPNs have been used in many domains (manufacturing, workflow management, distributed computing, embedded systems) and are supported by CPN Tools [13], a powerful framework for modeling, functional verification (several techniques) and performance analysis (stochastic discrete-event simulation). This makes CPNs very suitable for Octopus.

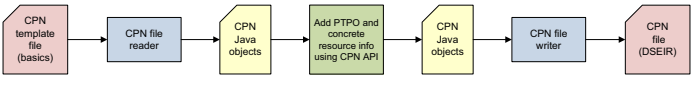


Fig. 6. Translating DSEIR specifications to CPN models

The interface between the DSEIR library and CPN Tools is realized in the DSEIR2CPN module. Any DSEIR specification can be converted. Several special cases are recognized to optimize the translated model in terms of simulation time. For maintainability and extensibility, the conversion uses a CPN template file containing the basic structure of the CPN model, high-level dynamics of the resource handler and monitors for producing simulation output. The information from DSEIR is added to this template, generating an executable CPN model. To allow reading and writing CPN models in Octopus, we built a Java API for a reasonably large class of CPN models. Fig. 6 shows the conversion process.

Fig. 7 shows the generated CPN model for our example, with a manually improved graphical layout to improve presentation. We do not go into details of the transformation, but rather briefly explain the generated model. The SYS CPN page shows that, at the highest level, the system consists of a PTPO (page

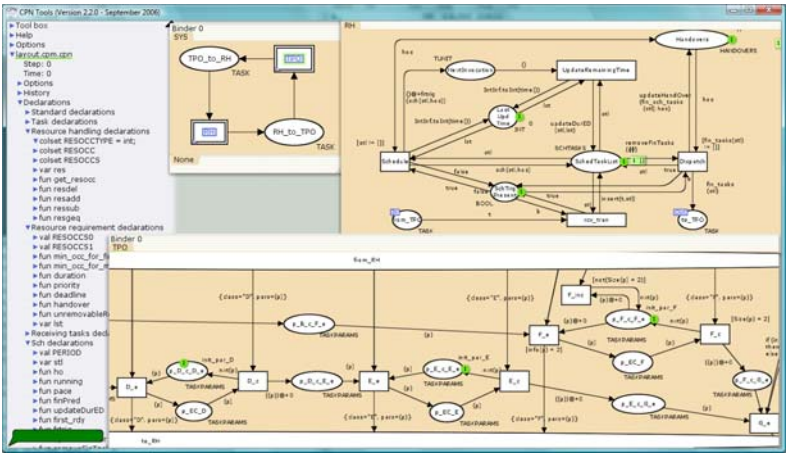


Fig. 7. CPN model obtained from the DSEIR specification of the running example

TPO), a resource handler (page RH) and the interface between them (places `TPO_to_RH` and `RH_to_TPO`). In the picture only the translations of tasks E and F and their connections to B, D, and G are shown. Tasks are split into an *enabling* and *completion* event, modeled as transitions with a place in between (`E_e`, `E_c`, and `p_EC_E` for task E); these transitions are connected to the appropriate interface places `to_RH` and `from_RH`. Parameter-scope restrictions are modeled as guards (only task F has a guard – predicate `Size`). Precedences are captured by a place between the two corresponding transitions of these tasks (place `p_D_c_E_e` for the CE precedence between D and E). A special `init` transition (not in the picture) is always the first transition to fire in a model. Its main purpose is to take a sample for any stochastic property in the PTPO (the workload in our example). The TPO page is the only part of the CPN model that has to be completely generated from DSEIR. The RH page is to a large extent generic for every DSEIR model, with many parts predefined in the template. Most of the scheduling dynamics is hidden in the CPN ML functions on the left.

6.2 Transforming DSEIR Models to Timed Automata

Timed Automata (TA) are a suitable formalism to capture at least a subset of the systems that DSEIR can express. Powerful tools, the model checker Uppaal [2] being one of them, are available for the analysis of TA. Uppaal's powerful analysis engine and relatively mature modeling support (ease of use, extensiveness of the modeling language) make it very suitable for integration in Octopus.

The DSEIR2Uppaal module implements the translation of DSEIR to Uppaal's input language. It unfolds the PTPO to a concrete task partial order without parameters, which is possible for finite parameter ranges. Each task instance is modeled by a separate TA. The TA broadcast their start and end events via *channels*. Each TA has a number of *waiting* locations, and the *enabled*, *active* and *done* locations. Location *enabled* indicates that a task is schedulable for

execution. Fig. 8 shows the TA for the first instance of task G of our example. It has two *waiting* locations, as G must wait for tasks E and F. When G is enabled, it can actually start if it can take all required resources, which is modeled by the guard *canClaim*(). If it starts, it broadcasts its start over the *started*0 channel, takes the resources via the *claim*() function, and resets its *x_cpu* clock. The transition from *active* to *done* then happens after exactly *time*() time units. The TA broadcasts that it is finished, and releases the resources.

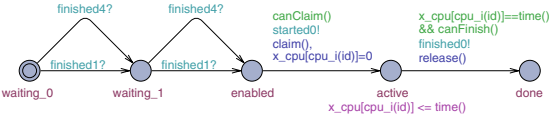


Fig. 8. The Uppaal timed automaton specification for the first instance of task G

A resource that is not shared is modeled by an integer variable that represents the remaining capacity and one or more clocks. These resources are claimed by decreasing the variable (function *claim*() in Fig. 8), and released by increasing the variable (function *release*()). If a task TA has claimed a resource, then it can use the clock of that resource for its timing. The TA in Fig. 8 uses one of the *x_cpu* clocks. Shared resources are modeled by a set of TA that implement preemption and redistribution of the resource amounts when a task starts or stops using the resource. Preemption cannot be modeled exactly with Uppaal, but it can be approximated with arbitrary precision [12]. Unfortunately, this technique fragments the symbolic representation of time that Uppaal uses and will generally have a negative effect on the performance of the analysis.

As in the CPN case, special cases of the transformation may be recognized. If no two instances of the same task can run concurrently, it is not necessary to unfold the PTPO. All instances of one task can then be captured in a single, iterative TA. This limits the number of automata in the model, thereby improving performance and reducing the size (in terms of bytes of memory) of a state.

7 Case Studies

Our first experimental evaluations with the Octopus toolset involve the running example used throughout this paper and two printer data path designs.

7.1 The Running Example

We use the combined strengths of CPN Tools and Uppaal, i.e., (stochastic) simulation and schedule optimization, to solve the DSE problem of Section 3.

Problem refinement: From the expected CPU workload of 5.7 per data object and the task E execution time of 6 time units, we conclude that 1/6 objects/time unit is an upper bound for throughput. The B-G subsystem allows this throughput if memory M_2 is sufficiently large. Due to incidental peak workloads on the CPU, this bound however can never be reached. It can be approached arbitrarily

close though with a sufficiently large M_1 memory. We decide to aim for memory sizes that allow a throughput within 2% of the upper bound. We also want a simple scheduling policy that provides this throughput. Because of the stochastic nature of the system, we are satisfied if repeated simulations show that the desired throughput is achieved with a 99% confidence.

Initial evaluation: The DSEIR model used throughout Section 5 (without task priorities, because the specification does not give priorities) is our initial model. Memories M_1 and M_2 are both set to the sufficiently large value of 200. Input streams, sampled from the distribution in Fig. 2, have 2000 objects.

Initial CPN Tools simulations show deadlocks. These are caused by the memory allocation strategy, and occur both on M_1 and on M_2. Both the A-B and the B-G subsystems are pipelines. If the heads of those pipelines work ahead too far and claim all the memory, tasks further down the pipeline are blocked.

The deadlocks occur independent of the memory sizes. We therefore change the memory allocation, switching from task-based allocation to object-based allocation. In this strategy, the first task needing a memory resource (A for M_1, B for M_2) claims the maximum amount needed in the pipeline and upon termination hands over to its successor the maximum amount needed in the remaining pipeline, releasing the rest (if any). This handover and release strategy is adopted by all tasks in the pipeline. This strategy is deadlock-free by construction.

M_2 optimization: The next steps further investigate the B-G subsystem, to optimize M_2. At the end, we then plan to reduce M_1 as far as possible. We start with a binary search using CPN Tools simulations to determine an initial M_2 bound with the object-based memory allocation. We further prioritize the execution of tasks further down the pipeline over tasks earlier in the pipeline. This is not necessary but it is expected to give better performance. It turns out that an M_2 size of 120 is needed for the desired throughput.

We then want to use Uppaal to investigate whether it is possible to optimize the B-G subsystem and further reduce the M_2 size by smart scheduling. For a sufficiently large M_1, we may assume that B always has data to execute. Removing task A and memory M_1 and conservatively (from the resource usage perspective) assuming that F is always executed then removes all stochastic behavior in the model, allowing the use of Uppaal. To maximize scheduling freedom, we remove task priorities and revert back to task-based memory allocation (which is more efficient than object-based allocation). We use an observer TA to monitor throughput. A binary search for M_2 with the upper bound from the simulations shows that a size of 110 is needed to allow an execution with the optimal throughput (which is 1/6 objects/time unit for this subsystem).

Given that a throughput-optimal execution exists for an M_2 size smaller than the earlier bound of 120, we want to find simple scheduling rules that provide an optimal throughput within this 110 bound. Not all executions are throughput-optimal. Some, for example, still lead to deadlock; as before, tasks early in the pipeline work too far ahead of tasks later in the pipeline. We decide to investigate scheduling rules $XY(k)$ that constrain the difference between the execution counts of tasks Y and X by integer k. For example, $GB(3)$ states that $B(p+3)$ should not become enabled before $G(p)$ has completed. These rules can be

integrated into the PTPO via CE precedences. We investigate rules $BG(k)$, which limit how far tasks early in the pipeline can work ahead. We further investigate rule $DF(0)$, which guarantees that F becomes only enabled after completion of D on the same object. This disallows the simultaneous execution of memory-expensive tasks D and F, exploiting the fact that F is not in the time-critical path of the application. Inspection of optimal schedules obtained with Uppaal suggests that rule $GF(2)$ would be a possible alternative for $DF(0)$. $GF(2)$ has a similar effect but is less restrictive.

We combine the scheduling rules with *greedy scheduling*, without priorities, that non-deterministically resolves conflicting memory claims. It turns out that $GB(3)$ with $GF(2)$ is the best combination. With an M_2 size of 110, it *guarantees* an optimal throughput for all executions. $GB(4)$ and $GB(5)$, both with $GF(2)$, provide good alternatives, guaranteeing an optimal throughput when M_2 is 120 and 130, respectively. When using the *task priorities* introduced earlier for simulations both combinations give optimal throughput also with an M_2 size of 110. These alternatives are of interest because they are less sensitive to intermittent stalls of task B, which may occur in the full system due to peak workloads for task A. $GF(2)$, which we would not have considered without analyzing the Uppaal results, outperforms $DF(0)$.

Final optimization: For the final optimization step, we take the full system with object-based memory allocation for the A-B subsystem and task-based memory allocation for the B-G subsystem. We investigate the three mentioned combinations of scheduling rules, with greedy, priority-based scheduling (with the mentioned priority scheme). The full system should achieve a throughput of at most 2% below the bound of 1/6 objects/time unit. CPN Tools simulations indicate that the $GB(5)$-$GF(2)$ combination performs best. An M_1 size of 110 suffices, and due to the slightly relaxed throughput constraint and the fact that F is not always executed, the size of M_2 can be further reduced to 100. Thus, the final result of the DSE is that M_1 and M_2 sizes need to be 110 and 100.

Concluding remarks: The unique selling point of a model checker is its ability to exhaustively explore the state space of a model and check whether desired properties hold. In this case study, Uppaal has been used to prove that *any greedy scheduler* in combination with various scheduling rules and memory sizes *guarantees* the required throughput. Such results are hard if not impossible to obtain with simulators. On the other hand, Uppaal cannot handle stochastic behavior, and it does not scale to very large models. With limited input streams for the Uppaal analyses, all simulation and analysis experiments performed for this DSE take at most a few minutes on normal laptops. The example DSE illustrates that CPN Tools and Uppaal may complement each other. The automatic translations to CPN Tools and Uppaal from common DSEIR specifications made experimenting very easy.

7.2 Modeling Printer Data Paths

To evaluate the expressive power of DSEIR, we did two case studies involving digital printer data paths of Océ Technologies, based in Venlo, the Netherlands.

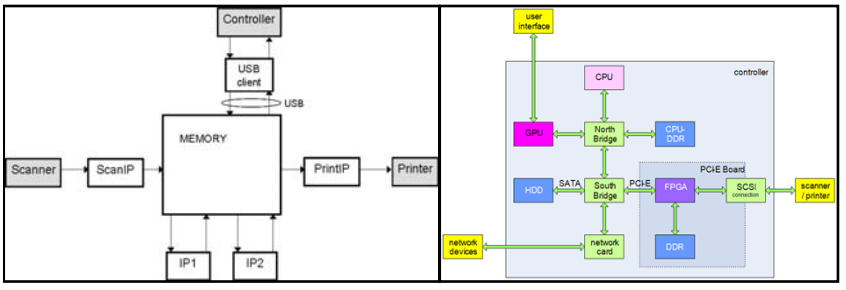


Fig. 9. The platforms in the printer case studies

The left picture in Fig. 9 is an abstracted version of an FPGA-based platform. It is used in a printer that supports use cases such as printing, copying, scanning, scan-to-email, and print-from-store. The machine can be accessed locally, through the scanner and the controller, and remotely, through the controller. The use cases all use different components in the platform. Various tasks can execute in parallel. Resources like the memory and the associated memory bus, as well as the USB are shared among tasks and among print and scan jobs running in parallel. The available USB bandwidth moreover dynamically fluctuates depending on whether it is used in one or in two directions simultaneously. Finding the fastest schedule for a sequence of jobs on the sketched platform is non-trivial. In [12], we modeled and analyzed this system with Uppaal. A novelty introduced in [12] was the already mentioned discrete approximation of the dynamic USB behavior. For the current paper, we modeled the system in DSEIR, automatically generated Uppaal models, and compared the results with the results obtained from the handcrafted models used for [12]. We achieved the same results, with similar analysis times (typically in the order of a few minutes).

The right picture in Fig. 9 shows a heterogeneous multiprocessor platform that combines one or more CPUs (running windows) with a GPU, one or more Harddisks (HDDs), and an FPGA. Because of heterogeneity and the use of general CPUs, the platform is more challenging than the platform of the first case study. We modeled the print use case for this platform in CPN Tools, to analyse the achieved throughput in images per second under dynamically fluctuating workloads, and to find out the appropriate buffer sizes between components. We later modeled the print use case on this platform in DSEIR and automatically derived a CPN model. Also in this case, we obtained matching results.

Together, the two case studies show that DSEIR is sufficiently expressive to capture a variety of realistic systems.

8 Conclusions

We have presented our ideas about model-driven design-space exploration (DSE) for embedded systems, and a first version of the Octopus DSE toolset based on these ideas. This toolset is organized around an intermediate representation, DSEIR, that is specifically designed to support DSE. It allows the independent

specification of applications, platforms, and mappings in a compact and precise way. The notion of a Parameterized Timed Partial Order for capturing applications is a novel element of DSEIR. The toolset is open and extensible. It aims to integrate existing formal analysis and simulation tools in the DSE process, to leverage their combined strengths. Our case studies show that combining tools is meaningful and useful. The toolset, DSEIR in particular, provides model consistency between analyses with different tools and easy experimentation. It allows easy reuse of tools among application domains by providing an intermediate between domain-specific modeling abstractions and formal analysis tools.

In future work, we plan to evaluate the toolset in other application domains. The current work mostly focuses on connecting tools, and on modeling and analyzing design alternatives. Future work will also focus on capturing experiment design and on automating (parts of) the DSE process itself, among others schedule optimization. We plan to develop a structured DSE method in which tools complement each other. New analysis tools will be added when the need or opportunity arises. Other relevant additions are a model repository with support for model versioning, and decision support. An interesting extension beyond the core DSE process is code generation support. The latest information about the Octopus toolset is available through `dse.esi.nl`.

References

1. Balarin, F., et al.: Hardware-Software Co-design of Embedded Systems: The POLIS Approach. Kluwer, Dordrecht (1997)
2. Behrmann, G., et al.: Uppaal 4.0. In: Proc. QEST, pp. 125–126. IEEE, Los Alamitos (2006)
3. Bleuler, S., et al.: PISA – A Platform and Programming Language Independent Interface for Search Algorithms. In: Fonseca, C.M., Fleming, P.J., Zitzler, E., Deb, K., Thiele, L. (eds.) EMO 2003. LNCS, vol. 2632, pp. 494–508. Springer, Heidelberg (2003)
4. Brekling, A.W., et al.: Models and Formal Verification of Multiprocessor System-on-Chips. J. Log. Algebr. Program. 77(1-2), 1–19 (2008)
5. Davare, A., et al.: A Next-Generation Design Framework for Platform-based Design. In: Proc. DVCon 2007 (February 2007)
6. David, A., et al.: Model-based Framework for Schedulability Analysis Using Uppaal 4.1. In: Model-based Design for Embedded Systems, pp. 121–143. Taylor & Francis, Abington (2009)
7. CoFluent Design. CoFluent Studio (2010), `http://www.cofluentdesign.com`
8. Esterel, Scade (2010), `http://www.esterel-technologies.com/products/scade-suite`
9. Gries, M.: Methods for Evaluating and Covering the Design Space during Early Design Development. Integration, the VLSI Journal 38, 131–183 (2004)
10. Hamann, A., et al.: A Framework for Modular Analysis and Exploration of Heterogeneous Embedded Systems. Real-Time Systems 33, 101–137 (2006)
11. Hennessy, M., Milner, R.: On Observing Nondeterminism and Concurrency. In: de Bakker, J.W., van Leeuwen, J. (eds.) ICALP 1980. LNCS, vol. 85, pp. 299–309. Springer, Heidelberg (1980)
12. Igna, G., et al.: Formal Modeling and Scheduling of Data Paths of Digital Document Printers. In: Cassez, F., Jard, C. (eds.) FORMATS 2008. LNCS, vol. 5215, pp. 170–187. Springer, Heidelberg (2008)

13. Jensen, K., et al.: Coloured Petri Nets and CPN Tools for Modelling and Validation of Concurrent Systems. STTT 9(3-4) (2007)
14. Jensen, K., Kristensen, L.M.: Coloured Petri Nets. Springer, Heidelberg (2009)
15. Keinert, J., et al.: SystemCoDesigner – An Automatic ESL Synthesis Approach by Design Space Exploration and Behavioral Synthesis for Streaming Applications. ACM Trans. Design Automation of Electronic Systems 14, Art. No. 1 (2009)
16. Kienhuis, B., et al.: An Approach for Quantitative Analysis of Application-specific Dataflow Architectures. In: Proc. ASAP 1997, pp. 338–34. IEEE, Los Alamitos (1997)
17. Ledeczi, A., et al.: Modeling Methodology for Integrated Simulation of Embedded Systems. ACM Trans. Model. Comput. Simul. 13(1), 82–103 (2003)
18. Lee, I., et al.: Resources in Process Algebra. J. Log. Algebr. Progr. 72, 98–122 (2007)
19. Lukasiewycz, M., et al.: Opt4J: Meta-heuristic Optimization Framework for Java. opt4j.sourceforge.net (2010)
20. Mousavi, M.R., et al.: PARS: A Process Algebra with Resources and Schedulers. In: Larsen, K.G., Niebert, P. (eds.) FORMATS 2003. LNCS, vol. 2791, pp. 134–150. Springer, Heidelberg (2004)
21. Palermo, G., et al.: Multi-objective Design Space Exploration of Embedded Systems. J. Embedded Computing 1, 305–316 (2005)
22. Pimentel, A.D.: The Artemis Workbench for System-Level Performance Evaluation of Embedded Systems. Int'l J. Embedded Systems 3(3), 181–196 (2008)
23. Sander, I., Jantsch, A.: System Modeling and Transformational Design Refinement in ForSyDe. IEEE T. Comput.-Aid. Design 23(1), 17–32 (2004)
24. MLDesign Technologies. MLDesigner (2010), http://www.mldesigner.com
25. Theelen, B.D., et al.: Software/Hardware Engineering with the Parallel Object-Oriented Specification Language. In: Proc. Memocode 2007, pp. 139–148. IEEE, Los Alamitos (2007)
26. Trcka, N., et al.: Parameterized Timed Partial Orders with Resources: Formal Definition and Semantics. Tech. Rep. ESR-2010-01, Eindhoven Univ. of Tech. (2010)
27. Uppaal Web Site, Web Help (2010), http://www.uppaal.com
28. Viskic, I., et al.: Design Exploration and Automatic Generation of MPSoC Platform TLMs from Kahn Process Network Applications. In: Proc. LCTES, pp. 77–84. ACM, New York (2010)

Contract-Based Slicing

Daniela da Cruz, Pedro Rangel Henriques, and Jorge Sousa Pinto

Departamento de Informática / CCTC
Universidade do Minho, 4710-057 Braga, Portugal
{danieladacruz,prh,jsp}@di.uminho.pt

Abstract. In the last years, the concern with the correctness of programs has been leading programmers to enrich their programs with annotations following the principles of *design-by-contract*, in order to be able to guarantee their correct behaviour and to facilitate reuse of verified components without having to reconstruct proofs of correctness.

In this paper we adapt the idea of *specification-based slicing* to the scope of (contract-based) program verification systems and behaviour specification languages. In this direction, we introduce the notion of *contract-based slice* of a program and show how any specification-based slicing algorithm can be used as the basis for a contract-based slicing algorithm.

1 Introduction

Program slicing plays an important role in program comprehension, enabling engineers to focus on just a relevant part (with respect to a given criterion) of a program. After Weiser's pioneering work [13], many researchers have searched for more effective or more powerful slicing techniques; since then, many application areas have been identified, including program debugging, software maintenance, software reuse, and so on. See [14] for a fairly recent survey of the area.

Many studies have proposed the use of slicing for software testing. In the context of complex applications, which are by their very nature, size and architecture difficult to comprehend and test, slicing may be an invaluable help when a certification process has to be carried out.

On the other hand a strong demand for formal methods that help programmers developing correct programs has been present in software engineering for some time now. The *Design by Contract* (DbC) approach to software development [12] facilitates modular verification and certified code reuse, and has become a standard approach to the design of architecturally complex systems. The contract for a component (a procedure) can be regarded as a form of enriched software documentation that fully specifies the behavior of that component.

The development and broad adoption of annotation languages for the most popular programming languages reinforces the importance of using DbC principles in the development of programs. These include for instance the Java Modeling Language (JML) [4]; Spec# [2], a formal language for C# API contracts, and the SPARK [1] subset of Ada.

T. Margaria and B. Steffen (Eds.): ISoLA 2010, Part I, LNCS 6415, pp. 106–120, 2010.

Traditional program slicing is based on control and data dependency analyses, but forms of slicing based on logical assertions have also been studied for over 10 years now, which combine slicing techniques with program verification, to identify synergies and take advantage of good practices on both sides. Comuzzi introduced the concept of *p-slice* [7], which is a slice calculated with respect to the validity of a given postcondition Q. The idea is that all program statements that are not required for the validity of Q upon termination are removed from the program (this only makes sense if Q holds as postcondition for the initial program). Canfora and colleagues used preconditions in their *conditioned slicing* technique [5] as a means to specify a set of initial states for computing a slice, resulting in a mixed form halfway between static and dynamic slicing. Preconditions and postconditions were combined by Chung and colleagues [6] to calculate what they called *specification-based slices*. Finally, *Fox et al* [9] introduced the *backward conditioning* technique, based on symbolic execution, to slice statements which, when executed, always lead to the negation of a given postcondition. The goal here is to use slicing as an aid in the verification of programs, in particular to find bugs.

In this paper we consider programs as sets of contract-annotated procedures, and study notions of assertion-based slicing for such programs with contracts. Specifically, we introduce the concept of *contract-based slice* of a program. Given any specification-based slicing algorithm (working at the level of commands), a contract-based slice can be calculated by slicing the code of each individual procedure independently with respect to its contract (which we call an *open* slice), or taking into consideration the calling contexts of each procedure inside a program (which we call a *closed* slice). We study both notions and then go on to introduce a more general notion of contract-based slice, which encompasses both open and closed slices as extreme cases. We remark that although the language used in this paper to illustrate our ideas is very simple, the principles and algorithms presented here scale up to realistic languages.

Structure of the Paper. Section 2 introduces a simple imperative language with annotated mutually recursive procedures, and sets up a verification conditions generator (VCGen) for that language. Section 3 then formalizes the notion of specification-based slice for the language. Sections 4 and 5 introduce contract-based slicing in their more specific (open and closed) and general forms respectively; section 6 then shows how a contract-based slicing algorithm (working at the inter-precedural level) can be synthesized from any desired specification-based slicing algorithm (working at the intra-precedural level). Section 7 illustrates our ideas through an example, and Section 8 concludes the paper.

2 Foundations: Verification Conditions and Specification-Based Slicing

To illustrate our ideas we use a simple programming language. Its syntax is defined in Figure 1, where x and $\mathbf{p}$ range over sets of variables and procedure

$$
\begin{array}{lll}
\textbf{Exp[int]} & \ni e & ::= \ldots \mid -1 \mid 0 \mid 1 \mid \ldots \mid x \mid x\tilde{} \mid \mathit{result} \mid -e \mid e + e \mid e - e \mid e * e \\
& & \mid \; e\,\texttt{div}\,e \mid e\,\texttt{mod}\,e \\
\textbf{Exp[bool]} & \ni b & ::= \texttt{true} \mid \texttt{false} \mid e == e \mid e < e \mid e <= e \mid e > e \mid e >= e \mid e \mathrel{!}= e \\
& & \mid \; b \mathbin{\&\&} b \mid b \parallel b \mid \mathrel{!} b \\
\textbf{Comm} & \ni C & ::= \textbf{skip} \mid C\,;\,C \mid x := e \mid \textbf{if}\ b\ \textbf{then}\ C\ \textbf{else}\ C \mid \textbf{while}\ b\ \textbf{do}\ \{A\}\,C \mid \textbf{call}\,\mathbf{p} \\
\textbf{Assert} & \ni A & ::= \texttt{true} \mid \texttt{false} \mid e == e \mid e < e \mid e <= e \mid e > e \mid e >= e \mid e \mathrel{!}= e \\
& & \mid \; \mathrel{!} A \mid A \mathbin{\&\&} A \mid A \parallel A \mid A \rightarrow A \mid \texttt{Forall}\,x.\,A \mid \texttt{Exists}\,x.\,A \\
\textbf{Proc} & \ni \Phi & ::= \textbf{pre}\ A\ \textbf{post}\ A\ \textbf{proc}\ \mathbf{p} = C
\end{array}
$$

C	$\mathsf{wp}(C, Q)$	$\mathsf{VC}(C, Q)$
skip	Q	$\emptyset$
$C_1\,;\,C_2$	$\mathsf{wp}(C_1, \mathsf{wp}(C_2, Q))$	$\mathsf{VC}(C_1, \mathsf{wp}(C_2, Q))$ $\cup\ \mathsf{VC}(C_2, Q)$
$x := e$	$Q[e/x]$	$\emptyset$
if b **then** C_t **else** C_f	$(b \rightarrow \mathsf{wp}(C_t, Q)) \mathbin{\&\&} (\mathrel{!} b \rightarrow \mathsf{wp}(C_f, Q))$	$\mathsf{VC}(C_t, Q)\ \cup\ \mathsf{VC}(C_f, Q)$
while b **do** $\{I\}\,C$	I	$\{(I \mathbin{\&\&} b) \rightarrow \mathsf{wp}(C, I),$ $(I \mathbin{\&\&} \mathrel{!} b) \rightarrow Q\}$ $\cup\ \mathsf{VC}(C, I)$
call $\mathbf{p}$	$\texttt{Forall}\,\overline{x_f}.$ $\big(\mathbf{pre}(\mathbf{p}) \rightarrow \mathbf{post}(\mathbf{p})\ \big[\overline{x}/\overline{x\tilde{}}, \overline{x_f}/\overline{x}\big]\big)$ $\rightarrow Q[\overline{x_f}/\overline{x}]$ with $\overline{x} = \mathcal{N}(\mathbf{post}(\mathbf{p})) \cup \mathcal{N}(Q)$	$\emptyset$

- The operator $\mathcal{N}(\cdot)$ returns a sequence of the variables occurring free in its argument assertion.
- Given a sequence of variables $\overline{x} = x_1, \ldots, x_n$, we let $\overline{x_f} = x_{1f}, \ldots, x_{nf}$ and $\overline{x\tilde{}} = x_1\tilde{}, \ldots, x_n\tilde{}$
- The expression $t[\overline{e}/\overline{x}]$, with $\overline{x} = x_1, \ldots, x_n$ and $\overline{e} = e_1, \ldots, e_n$ denotes the parallel substitution $t[e_1/x_1, \ldots, e_n/x_n]$

Fig. 1. Abstract syntax of programming language with annotations and VCGen rules

names respectively. A program is a non-empty set of mutually recursive procedure definitions that share a set of global variables (note that this is also an appropriate model for classes in an object-oriented language, whose methods operate on instance attributes). Operationally, an entry point would have to be defined for each such program, but that is not important for our current purpose. For the sake of simplicity we will consider only *parameterless procedures*

that work exclusively on global variables, used for input and output, but the ideas presented here can be easily adapted to cope with parameters passed by value or reference, as well as return values.

Each procedure consists of a body of code, annotated with a precondition and a postcondition that form the procedure's specification, or *contract*. The body may additionally be annotated with loop invariants. Occurrences of variables in the precondition and postcondition of a procedure refer to their values in the pre-state and post-state of execution of the procedure respectively; furthermore the postcondition may use the syntax $x\tilde{}$ to refer to the value of variable x in the pre-state (this is inspired by the SPARK syntax). For each program variable x, $x\tilde{}$ is a special variable that can only occur in postconditions of procedures; the use of *auxiliary variables* (that occur in assertions only, not in code) is forbidden.

C-like syntax is used for expressions; the language of annotated assertions (invariants and contracts) extends boolean expressions with first-order quantification. Note that defining the syntax of assertions as a superset of boolean expressions is customary in specification languages based on contracts, as used by verification toolsets for realistic programming languages such as the SPARK toolset [1]. This clearly facilitates the task of programmers when annotating code with contracts.

A *program* is well-defined if all procedures adhere to the above principles, and moreover all defined procedure names are unique and the program is closed with respect to procedure invocation. We will write $\mathcal{P}(\Pi)$ for the set of names of procedures defined in Π. The operators $\mathbf{pre}(\cdot)$, $\mathbf{post}(\cdot)$, and $\mathbf{body}(\cdot)$ return a routine's precondition, postcondition, and body command, respectively, i.e. given the procedure definition $\mathbf{pre}\ P\ \mathbf{post}\ Q\ \mathbf{proc}\ \mathbf{p} = C$ with $\mathbf{p} \in \mathcal{P}(\Pi)$, one has $\mathbf{pre}_\Pi(\mathbf{p}) = P$, $\mathbf{post}_\Pi(\mathbf{p}) = Q$, and $\mathbf{body}_\Pi(\mathbf{p}) = C$. The program name will be omitted when clear from context.

We adopt the common assumptions of modern program verification systems based on the use of a *verification conditions generator* (VCGen for short) that reads in a piece of code together with a specification, and produces a set of first-order proof obligations (*verification conditions*) whose validity implies the partial correctness of the code with respect to its specification. Recall that given a command C and assertions P and Q, the *Hoare triple* $\{P\}\,C\,\{Q\}$ is valid if Q holds after execution of C terminates, starting from an initial state in which P is true [10]. A set of first-order conditions $\mathsf{Verif}(\{P\}\,C\,\{Q\})$ whose validity is sufficient for this is given as

$$\mathsf{Verif}(\{P\}\,C\,\{Q\}) = \{P \rightarrow \mathsf{wp}(C, Q)\} \quad \cup \quad \mathsf{VC}(C, Q)$$

Where the functions $\mathsf{wp}(\cdot,\cdot)$ and $\mathsf{VC}(\cdot,\cdot)$ are defined in Figure 1. The first condition states that P is stronger than the weakest precondition that grants the validity of postcondition Q, and the remaining verification conditions ensure the adequacy of certain preconditions. For instance, the precondition of a loop

command can only be considered to be equal to the annotated invariant if this is indeed an invariant (whose preservation is ensured by a verification condition) and moreover it is sufficient to establish the truth of the required postcondition upon termination of the loop.

Definition 1 (Verif. Conditions of a Program). *For a program Π consisting of the set of procedures $\mathcal{P}(\Pi)$, the set of verification conditions* $\mathsf{Verif}(\Pi)$ *is*

$$\mathsf{Verif}(\Pi) \;=\; \bigcup_{\mathbf{p}\in\mathcal{P}(\Pi)} \mathsf{Verif}(\{\mathbf{pre}(\mathbf{p}) \;\&\&\; \overline{x} == \overline{x\tilde{}}\}\; \mathbf{body}(\mathbf{p})\; \{\mathbf{post}(\mathbf{p})\})$$

Note $\mathbf{pre}(\mathbf{p})$ is strengthened to allow for the use of the ˜ notation in postconditions. Let $\models A$ denote the validity of assertion A, and $\models S$, with S a set of assertions, the validity of all $A \in S$. This VCGen algorithm can be shown to be *sound* with respect to an operational semantics for the language, i.e. if $\models \mathsf{Verif}(\Pi)$ then for every $\mathbf{p} \in \mathcal{P}(\Pi)$ the triple $\{\mathbf{pre}(\mathbf{p})\}\, \mathbf{call}\, \mathbf{p}\, \{\mathbf{post}(\mathbf{p})\}$ is valid.

The intra-procedural (command-level) aspects of the VCGen are standard, but the inter-procedural aspects (program-level) are less well-known. We make the following remarks.

- Although this is somewhat hidden (unlike in the underlying program logic), the soundness of the VCGen is based on a *mutual recursion* principle, i.e. the proof of correctness of each routine assumes the correctness of all the routines in the program, including itself. If all verification conditions are valid, correctness is established simultaneously for the entire set of procedures in the program. This is the fundamental principle behind *design-by-contract*.
- The weakest precondition rule for procedure call takes care of what is usually known as the *adaptation* between the procedure's contract and the postcondition required in the present context. The difficulty of reasoning about procedure calls has to do with the need to refer, in a contract's postcondition, to the values that some variables had in the pre-state. We adapt to our context the rule proposed by Kleymann [11] as an extension to Hoare logic, and refer the reader to that paper for a historical discussion of approaches to adaptation.

Lemma 1. *Let $\models Q_1 \rightarrow Q_2$ with Q_1, Q_2 any two assertions. Then $\models \mathsf{wp}(C, Q_1) \rightarrow \mathsf{wp}(C, Q_2)$ and moreover $\models \mathsf{VC}(C, Q_1)$ implies $\models \mathsf{VC}(C, Q_2)$.*

3 Specification-Based Slicing

This section reviews the basic notions of specification-based slicing at the command level.

Informally, a command C' is a *specification-based slice* of C if it is a *portion* of C (a syntactic notion) and moreover C can be *refined* to C' with respect to a given specification (a semantic notion). We now give the formal definitions.

$$\frac{}{\mathbf{skip} \preceq C} \qquad \frac{C_1 \preceq C_2}{C_1\,;\,C \preceq C_2\,;\,C} \qquad \frac{C_1 \preceq C_2}{C\,;\,C_1 \preceq C\,;\,C_2}$$

$$\frac{C_1 \preceq C_2}{\mathbf{if}\ b\ \mathbf{then}\ C_1\ \mathbf{else}\ C \preceq \mathbf{if}\ b\ \mathbf{then}\ C_2\ \mathbf{else}\ C} \qquad \frac{C_1 \preceq C_2}{\mathbf{if}\ b\ \mathbf{then}\ C\ \mathbf{else}\ C_1 \preceq \mathbf{if}\ b\ \mathbf{then}\ C\ \mathbf{else}\ C_2}$$

$$\frac{C_1 \preceq C_2}{\mathbf{while}\ b\ \mathbf{do}\ \{I\}\ C_1 \preceq \mathbf{while}\ b\ \mathbf{do}\ \{I\}\ C_2}$$

Fig. 2. Definition of portion-of relation

Definition 2 (Portion-of relation). *The $\cdot \preceq \cdot$ relation is the transitive and reflexive closure of the binary relation generated by the set of rules given in Figure 2.*

Definition 3 (Specification-based slice [6]). *Let C, C' be commands and P, Q assertions such that $\models \mathsf{Verif}(\{P\}\,C\,\{Q\})$ holds, thus C is correct with respect to the specification (P, Q). C' is a slice of C with respect to (P, Q), written $C' \triangleleft_{(P,Q)} C$, if $C' \preceq C$ and $\models \mathsf{Verif}(\{P\}\,C'\,\{Q\})$.*

A *specification-based slicing algorithm* is any function slice that takes a command and a specification, and produces a slice of the command w.r.t. the specification, i.e. $\mathsf{slice}(C, P, Q) \triangleleft_{(P,Q)} C$.

Given program C correct with respect to the specification (P, Q), one wants to be able to identify portions of C that are still correct w.r.t (P, Q), i.e., portions in which some irrelevant statements (in the sense that they are not required for the program to be correct) are removed. Naturally, many such slices may exist, as well as methods for calculating them. These methods differ with respect to *efficacy* (being able to precisely identify all removable commands and effectively removing the highest possible number of such commands) and *efficiency* (how the execution time of the slicing algorithm varies with the number of lines of code). In [3] we explain the issues involved in detail, survey the existing algorithms, and propose improvements over those algorithms, concerning both precision and efficiency.

We remark that in practice one would not want to slice a block of code C with respect to its own specification (P, Q), unless maybe to confirm that it does not contain unnecessary code; but consider the situation in which one is asked to fulfill a weaker specification (P', Q'), i.e. $\models P' \rightarrow P$ and $\models Q \rightarrow Q'$. Then the code C can be reused, since it is necessarily also correct with respect to (P', Q'), but it may contain code that, while being necessary to satisfy (P, Q), is irrelevant with respect to satisfying (P', Q'). Thus in such a *specialization reuse* context, it makes sense to slice C with respect to the new specification to eliminate the unnecessary code.

A Specification-Based Slicing Algorithm

We have designed a specification-based slicing algorithm that improves on previous algorithms with respect to the number of statements removed from the programs. In fact we show in [3] that this algorithm produces the smallest slice of a program relative to a given specification (modulo an oracle for first-order proof obligations). The algorithm works on a *labelled control-flow graph*, whose edges are labelled with a pair of assertions corresponding to the strongest postcondition (calculated by propagating the specified precondition forward) and the weakest precondition (calculated by propagating the specified postcondition backward) at the program point represented by that edge. The graph is then extended by adding additional edges corresponding to subprograms S such that the strongest postcondition at the point immediately before S is stronger than the weakest precondition immediately after S (this implicative formula can be checked by an external proof tool). The resulting *slice graph* contains as subgraphs representations (in the form of labelled CFGs) of all the specification-based slices of the initial program with respect to its specification, and the smallest such slice can be calculated using standard graph algorithms.

Figure 3 shows an example slice graph for a program. Sliceable sequences are signaled by edges (and possibly **skip** nodes) added to the initial labeled CFG (shown as thick lines). Our online laboratory [8] implements this algorithm. The laboratory also allows for the visualization of these labelled control-flow graphs, which are useful as an aid in debugging, when the verification of a program fails.

4 Open / Closed Contract-Based Slicing

How can specification-based slicing be applied in the context of a multi-procedure program? Since a program is a set of procedures carrying their own contract specifications, it makes sense to investigate how the contract information can be used to produce useful slices at the level of individual procedures, and globally at the level of programs. A natural approach consists in simply slicing each procedure based on its own contract information. The idea is to eliminate all spurious code that may be present and does not contribute to that procedure's contract.

Definition 4 (Open Contract-based Slice). *Given programs Π, Π' such that $\models \mathsf{Verif}(\Pi)$ and $\mathcal{P}(\Pi) = \mathcal{P}(\Pi')$, we say that Π' is an open contract-based slice of Π, written $\Pi' \triangleleft_o \Pi$, if for every procedure $\mathbf{p} \in \mathcal{P}(\Pi)$ the following holds:* $\mathbf{pre}_{\Pi'}(\mathbf{p}) = \mathbf{pre}_{\Pi}(\mathbf{p})$*;* $\mathbf{post}_{\Pi'}(\mathbf{p}) = \mathbf{post}_{\Pi}(\mathbf{p})$*; and*

$$\mathbf{body}_{\Pi'}(\mathbf{p}) \triangleleft_{(\mathbf{pre}(\mathbf{p}) \;\&\&\; \overline{x}==\overline{x^{\sim}}, \mathbf{post}(\mathbf{p}))} \mathbf{body}_{\Pi}(\mathbf{p})$$

i.e. the body of each routine in Π' is a specification-based slice (with respect to its own annotated contract) of that routine in Π.

```
1 if (y > 0) then x := 100; x := x+50; x := x−100
2           else x := x−150; x := x+100; x := x+100
```

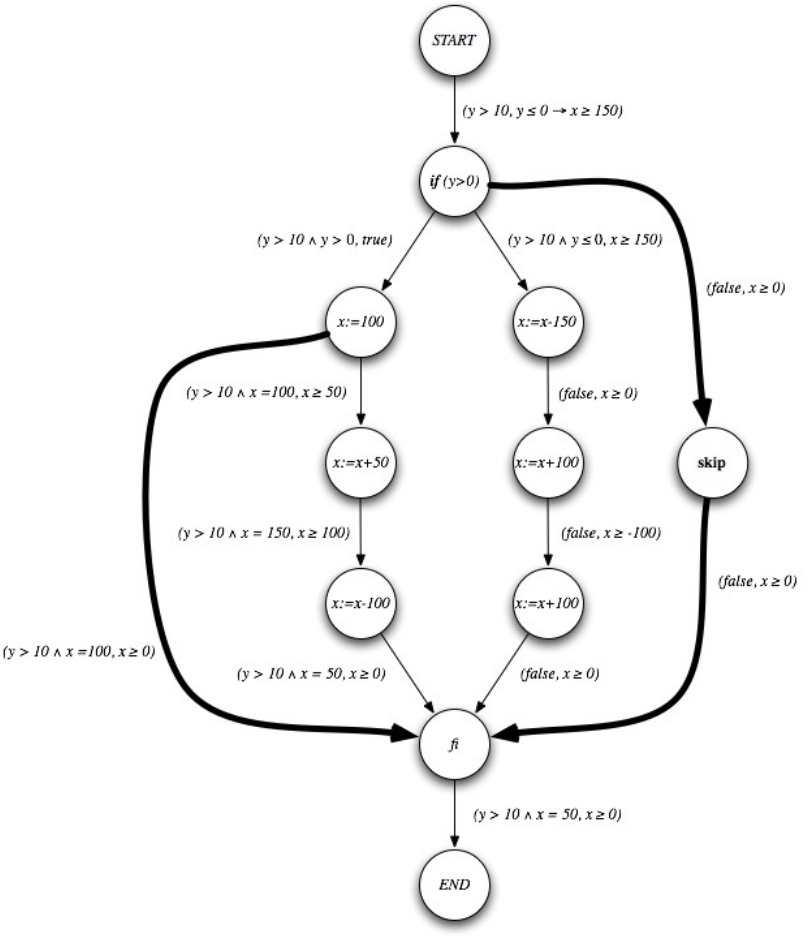


Fig. 3. Example program and its slice graph w.r.t. specification $(y > 10, x \geq 0)$

As expected, open contract-based slicing produces correct programs:

Proposition 1. *If* $\models \mathsf{Verif}(\Pi)$ *and* $\Pi' \lhd_o \Pi$ *then* $\models \mathsf{Verif}(\Pi')$.

Given some specification-based slicing algorithm $\mathsf{slice}(\cdot,\cdot,\cdot)$, program Π, and procedure $\mathbf{p} \in \mathcal{P}(\Pi)$, it is straightforward to lift it to an algorithm that calculates contract-based slices. Let

$$\begin{aligned}\mathsf{procslice_o}(\mathbf{p}) \doteq\ & \mathbf{pre}\ (\mathbf{pre}_\Pi(\mathbf{p}))\\ & \mathbf{post}\ (\mathbf{post}_\Pi(\mathbf{p})\\ & \mathbf{proc}\ \mathbf{p} = \mathsf{slice}(\mathbf{body}_\Pi(\mathbf{p}), \mathbf{pre}_\Pi(\mathbf{p}), \mathbf{post}_\Pi(\mathbf{p}))\end{aligned}$$

Then $\mathsf{progslice_o}(\Pi) \doteq \{\ \mathsf{procslice_o}(\mathbf{p}) \mid \mathbf{p} \in \mathcal{P}(\Pi)\ \}$.

Proposition 2. *For any program* Π *such that* $\models \mathsf{Verif}(\Pi)$, $\mathsf{progslice_o}(\Pi) \lhd_o \Pi$.

As was the case with specification-based slicing, one may want to calculate open contract-based slices just to make sure that a program (already proved correct) does not contain irrelevant code. This notion of slice of a program assumes that all the procedures are public and may be externally invoked – the program is *open*. But consider now the opposite case of a program whose procedures are only invoked by other procedures in the same program, which we thus call a *closed* program (this makes even more sense if one substitutes *class* for *program* and *method* for *procedure* in this reasoning). In this situation, since the set of callers of each procedure is known in advance (it is a subset of the procedures in the program), it is possible that *weakening* the procedures' contracts may still result in a correct program, as long as the assumptions required by each procedure call are all still respected. In other words the procedures may be doing more work than is actually required. Such a program may then be sliced in a more aggressive way, defined as follows.

Definition 5 (Closed Contract-based Slice). *Let* Π, Π' *be programs such that* $\models \mathsf{Verif}(\Pi)$ *and* $\mathcal{P}(\Pi) = \mathcal{P}(\Pi')$. Π' *is a* closed contract-based slice *of* Π, *written* $\Pi' \lhd_c \Pi$, *if* $\models \mathsf{Verif}(\Pi')$ *and additionally for every procedure* $\mathbf{p} \in \mathcal{P}(\Pi)$

1. $\models \mathbf{pre}_{\Pi'}(\mathbf{p}) \rightarrow \mathbf{pre}_\Pi(\mathbf{p})$;
2. $\models \mathbf{post}_\Pi(\mathbf{p}) \rightarrow \mathbf{post}_{\Pi'}(\mathbf{p})$; *and*
3. $\mathbf{body}_{\Pi'}(\mathbf{p}) \lhd_{(\mathbf{pre}_{\Pi'}(\mathbf{p})\ \&\&\ \overline{x}==\overline{x'}, \mathbf{post}_{\Pi'}(\mathbf{p}))} \mathbf{body}_\Pi(\mathbf{p})$

Note that in general weakening the contracts of some procedures in a correct program may result in an incorrect program, since the correctness of each procedure may depend on the assumed correctness of other procedures; thus the required condition $\models \mathsf{Verif}(\Pi')$ in the definition of closed contract-based slice.

5 Contract-Based Slicing: General Case

Clearly the notion of closed contract-based slicing admits trivial solutions: since all contracts can be weakened, any precondition (resp. postcondition) can be set

to *false* (resp. *true*), and thus any procedure body can be sliced to **skip**. A more interesting and realistic notion is obtained by fixing a subset of procedures of the program, whose contracts must be preserved. All other contracts may be weakened as long as the resulting program is still correct.

Definition 6 (Contract-based Slice). *Let Π, Π' be programs such that $\mathcal{P}(\Pi) = \mathcal{P}(\Pi')$ and $\mathcal{S} \subseteq \mathcal{P}(\Pi)$; Π' is a* contract-based slice *of Π, written $\Pi' \lhd_{\mathcal{S}} \Pi$, if the following all hold:*

1. $\models \mathsf{Verif}(\Pi')$.
2. *for every procedure* $\mathbf{p} \in \mathcal{S}$,
   - $\mathbf{pre}_{\Pi'}(\mathbf{p}) = \mathbf{pre}_{\Pi}(\mathbf{p})$ *and* $\mathbf{post}_{\Pi'}(\mathbf{p}) = \mathbf{post}_{\Pi}(\mathbf{p})$;
   - $\mathbf{body}_{\Pi'}(\mathbf{p}) \lhd_{(\mathbf{pre}(\mathbf{p}) \,\&\&\, \overline{x}==\overline{x^{\sim}}, \mathbf{post}(\mathbf{p}))} \mathbf{body}_{\Pi}(\mathbf{p})$
3. *for every procedure* $\mathbf{p} \in \mathcal{P}(\Pi) \setminus \mathcal{S}$,
   - $\models \mathbf{pre}_{\Pi'}(\mathbf{p}) \rightarrow \mathbf{pre}_{\Pi}(\mathbf{p})$;
   - $\models \mathbf{post}_{\Pi}(\mathbf{p}) \rightarrow \mathbf{post}_{\Pi'}(\mathbf{p})$; *and*
   - $\mathbf{body}_{\Pi'}(\mathbf{p}) \lhd_{(\mathbf{pre}_{\Pi'}(\mathbf{p}) \,\&\&\, \overline{x}==\overline{x^{\sim}}, \mathbf{post}_{\Pi'}(\mathbf{p}))} \mathbf{body}_{\Pi}(\mathbf{p})$

This notion is very adequate to model slicing when applied to code reuse. When program (or class) Π is reused, some of its procedures may not need to be public, since they will not be externally invoked (but they may be invoked by other, public procedures in the program). In this case the contracts of the private procedures may be weakened according to their usage inside the program, i.e. the actual required specification for each private procedure may be calculated from the set of internal calls, since it is guaranteed that no external calls will be made to private procedures. One may then want to reflect this in the annotated contracts, in order to produce a contract-based slice stripped of the redundant code. Private procedures whose contracts are not required internally may indeed see their bodies sliced to **skip**.

Note that even for closed programs this notion makes more sense. Since its procedures are not invoked externally from other programs' procedures, every closed program will naturally have a main procedure to be executed as an entry point, whose contract is fixed (cannot be weakened).

Finally, it is easy to see that both open and closed contract-based slicing are special cases of Definiton 6: $\Pi' \lhd_o \Pi \Leftrightarrow \Pi' \lhd_{\mathcal{P}(\Pi)} \Pi$ and $\Pi' \lhd_c \Pi \Leftrightarrow \Pi' \lhd_{\emptyset} \Pi$.

6 A Contract-Based Slicing Algorithm

Again any specification-based slicing algorithm (at the command level) can be used for calculating contract-based slices according to Definition 6. A contract-based slice of program Π can be calculated by analyzing Π in order to obtain information about the actual preconditions and postconditions that are required of each procedure call, and merging this information together. Specifically, we may calculate for each procedure the disjunction of all required preconditions and the conjunction of all required postconditions; this potentially results in a weaker

contract with respect to the annotated contract of the procedure, which can thus be used to slice that procedure.

To implement this idea for a given program Π we consider a preconditions table T_0 that associates to each procedure $\mathbf{p} \in \mathcal{P}(\Pi)$ a precondition, initialized with $T_0[\mathbf{p}] = \texttt{false}$, and a postconditions table T that associates to each procedure $\mathbf{p} \in \mathcal{P}(\Pi)$ a postcondition, initialized with $T[\mathbf{p}] = \texttt{true}$. The algorithm executes the following steps to produce $\mathsf{progslice}_{\mathsf{o}}(\Pi)$.

1. Calculate $\mathsf{Verif}(\Pi)$ and while doing so, for every invocation of the form $\mathsf{wp}(\mathbf{call}\,\mathbf{p}, Q)$ set $T[\mathbf{p}] := T[\mathbf{p}]\ \&\&\ Q$.
2. An alternative set of verification conditions based on *strongest postconditions* can be defined by rules that are fairly symmetric to those given in Figure 1 (omitted here). We calculate this set, and while doing so, for every invocation of the form $\mathsf{sp}(\mathbf{call}\,\mathbf{p}, P)$ set $T_0[\mathbf{p}] := T_0[\mathbf{p}] \parallel P$.
3. For $\mathbf{p} \in \mathcal{P}(\Pi)\backslash\mathcal{S}$ let

$$\begin{aligned}\mathsf{procslice}_{\mathcal{S}}(\mathbf{p}) \doteq\ & \mathbf{pre}\ T_0[\mathbf{p}] \\ & \mathbf{post}\ T[\mathbf{p}] \\ & \mathbf{proc}\ \mathbf{p} = \mathsf{slice}(\mathbf{body}(\mathbf{p}), T_0[\mathbf{p}], T[\mathbf{p}])\end{aligned}$$

4. Then $\mathsf{progslice}_{\mathcal{S}}(\Pi) = \{\ \mathsf{procslice}_{\mathsf{o}}(\mathbf{p}) \mid \mathbf{p} \in \mathcal{S}\ \} \cup \{\mathsf{procslice}_{\mathcal{S}}(\mathbf{p}) \mid \mathbf{p} \in \mathcal{P}(\Pi)\backslash\mathcal{S}\}$

Proposition 3. *For any program Π such that $\models \mathsf{Verif}(\Pi)$, $\mathsf{progslice}_{\mathcal{S}}(\Pi) \lhd_{\mathcal{S}} \Pi$.*

Note that step 1 (or 2) can be skipped, in which case $T[\mathbf{p}]$ (resp. $T_0[\mathbf{p}]$) should be set to $\mathbf{post}(\mathbf{p})$ (resp. $\mathbf{pre}(\mathbf{p})$), and slicing will be less aggressive, based on preconditions or postconditions only. Section 7 illustrates the application of this contract-based slicing algorithm (using only postconditions, i.e. with step 2 of the algorithm skipped) to a program containing a procedure $\mathbf{p}$ that calculates a number of different outputs given an input array; this program is being reused in a context in which no external calls are made to $\mathbf{p}$, and two internal procedures do call $\mathbf{p}$, but some of the information computed by $\mathbf{p}$ is not required by either of the calls. Then maybe some statements of $\mathbf{p}$ can be sliced off.

7 An Illustrative Example

In this section we intend to illustrate the concept of *contract-based slice* through an example. We have implemented a prototype program slicer[1] for a subset of Spec#, which includes many different specification-based slicing algorithms, and which we have used to calculate the slices shown below. Note that in an object-oriented language if we substitute the notions of class, method, and instance attribute for those of program, procedure, and global variables of our previous imperative setting, the ideas introduced earlier essentially apply without modifications.

[1] Available through a web-based interface from `http://gamaepl.di.uminho.pt/gamaslicer`

```
1  int summ, sumEven, productum, maximum, minimum;
2  public int[]! a = new int[100];
3  public int length = 100;
4
5  public void OpersArrays()
6    ensures summ == sum{int i in (0: length); a[i]};
7    ensures sumEven ==
8        sum{int i in (0: length); (((a[i] % 2)== 0)? a[i]:0)};
9    ensures productum == product{int i in (0:length); a[i]};
10   ensures maximum == max{int i in (0:length); a[i]};
11   ensures minimum == min{int i in (0:length); a[i]};
12 {
13   summ = 0; sumEven = 0; productum = 0;
14   maximum = Int32.MinValue; minimum = Int32.MaxValue;
15   for (int n = 0; n < length; n++)
16     invariant n <= length;
17     invariant summ == sum{int i in (0: n); a[i]};
18     invariant sumEven ==
19         sum{int i in (0: n); (((a[i] % 2)== 0)? a[i]:0)};
20     invariant productum == product{int i in (0: n); a[i]};
21     invariant maximum == max{int i in (0: n); a[i]};
22     invariant minimum == min{int i in (0: n); a[i]};
23   {
24     summ += a[n];
25     productum *= a[n];
26     if ((a[n] % 2) == 0) { sumEven += a[n]; }
27     if(a[n] > maximum) { maximum = a[n]; }
28     if(a[n] < minimum) { minimum = a[n]; }
29   }
30 }
```

Listing 1. Annotated method `OpersArrays`

Listings 1 and 2 contain extracts from a class Π containing an annotated method, called `OpersArrays`, which computes several array operations: the sum of all elements, the sum of all the even elements, the iterated product, and the maximum and minimum. Π contains two other methods `Average` and `Multiply`; the former computes the average of the elements belonging to the array, and the latter multiplies the minimum of the array by a parameter `y`. The code contains appropriate contracts for all methods, as well as loop invariants. All 3 methods are correct with respect to their contracts – we assume this has been established beforehand.

Moreover suppose that `OpersArrays` is a private method, i.e. in a given context it is known that it will not be invoked externally. As can be observed in Listing 3, the method `Average` calls the method `OpersArrays` and then uses the calculated value of the variable `summ`, and `Multiply` calls `OpersArrays` and then uses the value of variable `minimum`. Then it makes sense to calculate a contract-based slice of this class with $\mathcal{S} = \{\texttt{Average}, \texttt{Multiply}\}$, which will result in the method `OpersArrays` being stripped of the irrelevant code.

```
1  public double Average()
2    ensures result == (sum{int i in (0: length);a[i]})/length;
3   {
4      double average;
5      OpersArrays();
6      average = summ / length;
7      return average;
8   }
9
10 public int Multiply(int y)
11   requires y >= 0;
12   ensures result == (min{int i in (0: length); a[i]})*y;
13  {
14     OpersArrays();
15     int x = minimum, q = 0, i =0;
16     while(i <y )
17         invariant 0 <= i <= y;
18         invariant q == (i * x);
19     {
20         q = q + x;
21         i= i+1;
22     }
23     return q;
24  }
```

Listing 2. Annotated method `Average` and `Multiply`

In order to calculate $\mathsf{progslice}_{\mathcal{S}}(\Pi)$, T is first initialized as

$$T[\texttt{OpersArrays}] := \texttt{true}$$

After performing the first step of the algorithm we set

$$T[\texttt{OpersArrays}] := \texttt{true} \;\&\&\; Q_1 \;\&\&\; Q_2$$

where (calculations omitted)

$$Q_1 = \frac{summ}{length} == \frac{sum\{int\ i\ in(0 : length); a[i]\}}{length}$$

$$Q_2 = 0 \leq i \leq y \;\&\&\; q == i \times minimum$$

So, as the component `OpersArrays` is called twice in the context of the two previous components, the result is the weaker postcondition $\texttt{true} \;\&\&\; Q_1 \;\&\&\; Q_2$. The final step in the calculation of the slice using table T gives us

$$\mathsf{progslice}_{\mathsf{c}}(\Pi) \doteq \{\mathsf{procslice}_{\mathsf{o}}(\texttt{Average}), \mathsf{procslice}_{\mathsf{o}}(\texttt{Multiply}), \mathsf{procslice}_{\mathcal{S}}(\texttt{OpersArrays})\}$$

```
1   public void OpersArrays()
2     ensures summ == sum{int i in (0: length); a[i]};
3     ensures minimum == min{int i in (0:length); a[i]};
4   {
5     summ = 0;
6     minimum = Int32.MaxValue;
7     for (int n = 0; n < length; n++)
8       invariant n <= length;
9       invariant summ == sum{int i in (0: n); a[i]};
10      invariant minimum == min{int i in (0: n); a[i]};
11    {
12      summ += a[n];
13      if(a[n] < minimum) { minimum = a[n]; }
14    }
15  }
```

Listing 3. Sliced method `OpersArrays`

Calculating slice(**body**(`OpersArrays`), **pre**(`OpersArrays`), T[`OpersArrays`]) results in cutting the statements present in lines 14, 27, 28 and 29 (after removing from the invariant the predicates in lines 20–23, which contain only occurrences of variables that do not occur in T[`OpersArrays`]). The final sliced method is shown below. The other two methods remain unchanged.

8 Conclusion

We introduced notions of slicing for programs developed according to design-by-contract principles. The motivation was to bring to the inter-procedural level the power of specification-based slicing, which we believe has great application potential and will surely become more popular in coming years, profiting from advances in verification and automated proof technology. We believe that the ideas proposed can be useful in the traditional fields of application of slicing, such as source code analysis (to help in debugging or program comprehension); program maintenance (when more precision is required); certification of programs constructed by reusing annotated components; and the specialization of programs composed of fully annotated and certified procedures.

This work sets up a theoretical framework for slicing multi-procedure, contract-annotated programs, and is part of a bigger effort that includes a fundamental investigation of command-level (intra-procedural) specification-based slicing algorithms [3], as well as the development of an online laboratory for assertion-based slicing [8] that interacts with external automatic theorem provers for discharging slicing conditions. Our approach can be applied to program comprehension (as it makes easier to understand the process of slicing a program with contracts) and program reuse (it can be seen as a lever to integrate annotated components with other programs in an easier manner).

As future work we intend to continue with the development of GamaSlicer in order to perform tests with more examples as well as to confirm that our approach scales up.

Acknowledgment. This work was partially supported by projects RESCUE (FCT contract PTDC / EIA / 65862 / 2006) and CROSS (FCT contract PTDC / EIA-CCO / 108995 / 2008).

References

1. Barnes, J.: High Integrity Software: The SPARK Approach to Safety and Security, 1st edn. Addison Wesley, Reading (March 2003)
2. Barnett, M., Rustan, K., Leino, M., Schulte, W.: The Spec# programming system: An overview. In: Barthe, G., Burdy, L., Huisman, M., Lanet, J.-L., Muntean, T. (eds.) CASSIS 2004. LNCS, vol. 3362, pp. 49–69. Springer, Heidelberg (2005)
3. Barros, J., da Cruz, D., Henriques, P.R., Pinto, J.S.: Assertion-based Slicing and Slice Graphs. In: Fiadeiro, J.L., Gnesi, S. (eds.) Proceedings of the 8th IEEE International Conference on Software Engineering and Formal Methods (SEFM 2010) (2010)
4. Burdy, L., Cheon, Y., Cok, D., Ernst, M., Kiniry, J., Leavens, G., Leino, K., Poll, E.: An overview of jml tools and applications (2003)
5. Canfora, G., Cimitile, A., De Lucia, A.: Conditioned program slicing. Information and Software Technology 40(11-12), 595–608 (1998)
6. Chung, I.S., Lee, W.K., Yoon, G.S., Kwon, Y.R.: Program slicing based on specification. In: SAC 2001: Proceedings of the 2001 ACM Symposium on Applied Computing, pp. 605–609. ACM, New York (2001)
7. Comuzzi, J.J., Hart, J.M.: Program slicing using weakest preconditions. In: Gaudel, M.-C., Woodcock, J.C.P. (eds.) FME 1996. LNCS, vol. 1051, pp. 557–575. Springer, Heidelberg (1996)
8. da Cruz, D., Henriques, P.R., Pinto, J.S.: Gamaslicer: an Online Laboratory for Program Verification and Analysis. In: Proceedings of the 10th Workshop on Language Descriptions Tools and Applications, LDTA 2010 (2010)
9. Fox, C., Danicic, S., Harman, M., Hierons, R.M.: Backward conditioning: A new program specialisation technique and its application to program comprehension. In: IWPC, pp. 89–97. IEEE Computer Society, Los Alamitos (2001)
10. Hoare, C.A.R.: An axiomatic basis for computer programming. Communications of the ACM 12, 576–580 (1969)
11. Kleymann, T.: Hoare logic and auxiliary variables. Formal Aspects of Computing 11(5), 541–566 (1999)
12. Meyer, B.: Applying "design by contract". Computer 25(10), 40–51 (1992)
13. Weiser, M.: Program slicing. In: ICSE 1981: Proceedings of the 5th International Conference on Software Engineering, Piscataway, NJ, USA, pp. 439–449. IEEE Press, Los Alamitos (1981)
14. Xu, B., Qian, J., Zhang, X., Wu, Z., Chen, L.: A brief survey of program slicing. SIGSOFT Softw. Eng. Notes 30(2), 1–36 (2005)

Special Track on Worst Case Traversal Time (WCTT)

Anne Bouillard[1], Marc Boyer[2], Samarjit Chakraborty[3], Steven Martin[4], Jean-Luc Scharbarg[5], Giovanni Stea[6], and Eric Thierry[7]

[1] ENS Cachan, France
[2] ONERA, France
[3] Technische Universität München, Germany
[4] LRI, France
[5] IRIT, France
[6] University of Pisa, Italy
[7] ENS Lyon, France

Real-time systems are increasingly becoming communication intensive, where different functions are implemented using distributed real-time tasks mapped onto different physical systems (sensors, processors, and actuators). To ensure global correctness, one has to ensure the correctness of each task, schedulability of the tasks on each system, and finally also bound the communication time, i.e., the worst case (network) traversal time (WCTT).

Moreover, as systems grow bigger, connecting dozens to hundreds of systems, managing thousands of traffic flows, with each one interacting with the others, is becoming a challenging problem. This special track is devoted to formal methods on worst case traversal time in networks, with a special interest on scalable methods and on handling multi-hop networks.

Our conviction is that WCTT is an emerging area, made up of people from different communities, using different methods, and publishing in different conferences and/or journals.

People from several active research groups in this domain were contacted, and almost all of them have submitted a paper, despite the very short deadline. This reactivity confirms the desire of a community to find places to meet and discuss.

The aim of this track is to be the first meeting dedicated to WCTT, allowing us to capture a snapshot of this area, and also for fostering discussions on how this area should evolve.

T. Margaria and B. Steffen (Eds.): ISoLA 2010, Part I, LNCS 6415, p. 121, 2010.

The PEGASE Project: Precise and Scalable Temporal Analysis for Aerospace Communication Systems with Network Calculus*

Marc Boyer[1], Nicolas Navet[2], Xavier Olive[3], and Eric Thierry[4]

[1] ONERA, 2 av. E. Belin 31055 Tououse Cedex 4, France
Marc.Boyer@onera.fr
[2] INRIA / RealTime-at-Work (RTaW), 615 rue du Jardin Botanique, 54600 Villers-lès-Nancy, France
Nicolas.Navet@realtimeatwork.com
[3] Thales Alenia Space, Research Department, 100 bd du Midi, 06156 Cannes La Bocca Cedex
xavier.olive@thalesaleniaspace.com
[4] LIP, ENS Lyon, 46 allée d'Italie, 69364 Lyon Cedex 07, France
Eric.Thierry@ens-lyon.fr

Abstract. With the increase of critical data exchanges in embedded real-time systems, the computation of tight upper bounds on network traversal times is becoming a crucial industrial need especially in safety critical systems. To address this need, the French project PEGASE grouping academics and industrial partners from the aerospace field has been undertaken to improve some key aspects of the Network Calculus and its implementation.

1 Introduction

Critical real-time embedded systems (cars, aircrafts, spacecrafts) are nowadays made up of multiple computers communicating with each other. The real-time constraints typically associated with local applicative tasks now extend to the communication networks between sensors/actuators and computers, and between the computers themselves. Once a communication medium is shared, the time between sending and receiving a message depends not only on the technological constraints, but mainly on the interactions between the different streams of data sharing the media.

It is therefore necessary to have techniques to guarantee, in addition to local scheduling constraints, the worst case traversal time of the network (WCTT) and thus ensure a correct global real-time behaviour of the distributed applications/functions. If the temporal evaluation techniques used are too pessimistic, it leads to an over-dimensioning of the network which involves extra cost, weight and power consumption. In addition to being precise, these verification techniques must be scalable. For instance, in a modern aircraft, thousands of data

* This work has been partialy funded by French ANR agency under project id ANR-09-SEGI-009.

T. Margaria and B. Steffen (Eds.): ISoLA 2010, Part I, LNCS 6415, pp. 122–136, 2010.

streams share the network backbone and therefore the complexities of the algorithm should be at most polynomial.

In the French PEGASE project, we aim to improve the theory of Network Calculus [1,2], which has already been used to certify the AFDX network of the A380, and its algorithmic implementation in order to meet the scalability and tigthness requirements. To assess the gains achieved and the practicability of the software tool in an industrial context, 3 case-studies have been undertaken respectively on AFDX [3], SpaceWire [4] and a NoC.

Section 2 presents the industrial context of embedded real-time networks. Section 3 provides a recap of the main techniques that can be used to compute worst-case traversal times. The case-studies of the project are described in section 4. The last two sections of the paper gives an overview of the first results of the project: some theoretical results in section 5, and the current version of the tool in section 6.

2 Industrial Context

2.1 Wide-Scale Communicating Systems

As described in the introduction, modern real-time critical embedded systems are composed of dozens to hundreds of electronic equipments (including computers, smart sensors...), communicating through thousands of data flows. To guarantee the correctness of the system's functions, real-time behavior of each application on each equipment must be ensured (with schedulability analysis), but the temporal validity of the data consumed (their "freshness") must also be managed, that is to say, the delivery of the data must be guaranteed and the delay introduced by the network must be bounded.

2.2 Shared Resources: Homogeneous vs. Heterogeneous Flows

To reduce weight and integration costs, networks are shared resources. But this sharing could have different degrees. In a first industrial step, networks are shared by homogeneous flows: one network for control command flows (small data, small bandwidth, need of low latency), and another for mission data flows (large images in spacecraft without real time constraints but guarantee of delivery is needed). In a second step, the network can be shared by heterogeneous flows (control command *and* mission data). One can also distinguish critical and non critical flows (control command versus entertainment in aircraft).

Mixing critical and non-critical, control/command and mission-specific streams may lead to significant gains. It is the opportunity to have a single on-board data network providing thus some significant savings. Besides, the margins can be factorized, reducing de-facto the need of over-sizing the system to fulfill the dependability requirements. Some other elements speak also in favor of this heterogeneous data mix, and all permits to save some time and money in general. Nevertheless, the validation or certification of such network architecture requires providing for each flow a bound on data delivery, and the proof of segregation (i.e. no interaction) between the flows.

2.3 Mono-Segment vs. Multi-hop (Homogeneous or Heterogeneous)

The last point is the evolution from mono-segment to multi-hop networks. If the system is small enough, all the nodes can be connected to the same segment. There are numerous network technologies suited for this type of architecture, for example, MIL-STD-1553, CAN, TTP, FlexRay, etc, but each one has limits on the global bandwidth and the number of equipments that can be connected[1]. Therefore, a complete system could be too large for a single segment. In this case, a multi segments network must be built, interconnecting segments with gateways or switches. In this context, an end-to-end communication can be multi-hops.

Moreover, multi-hop communications can be homogeneous (with the same technologies at each segment, like the AFDX or SpaceWire technologies), or heterogeneous: a sensor could be connected to a low-bandwidth segment, interconnected to a system backbone, where nodes reading the sensor values are connected. The real-time communication is, in this case, multi-hop and heterogeneous.

2.4 Use of Formal Methods in the Development Process

If computing WCTT is an important issue in designing critical real-time systems, this is not the only one from a global industrial point of view: three others (at least) should be considered: design complexity, method stability and method simplicity.

Verification of a platform is the second half of a problem: building a configuration is the first half. From an industrial point of view, a system too hard to be (efficiently) configured can be rejected, even if there are methods to verify it. For example, priority levels are a way to ensure performances and segregation, but they are not an intrinsinc characteristic of an applicative data flow. It might be difficult to map, in an efficient manner, thousands of data-flows onto dozens or hundreds of priority levels.

There is also a trade-off between complexity of the verification and confidence: if a formal method or a model is too complex, it is prone to contain errors, or its implementation tool might have bugs. From the industrial point of view, the simplest method to guaranty/certify the WCTT will be the easiest to adopt.

3 Related Works

The need for formal models to compute end-to-end bounds on delays and buffer consumptions is relatively new in embedded systems, due to a change in the technologies used. Until recently, the embedded bus technologies were quite simple, leading to constant or at least easy to bound delays (ARINC429, MIL-STD-1553, CAN). But the introduction of new technologies (AFDX, SpaceWire), more efficient in some ways but also more complex, creates the need for new and/or improved analysis methods. These models must be correct, i.e. the computed

[1] Sometimes, such as for TTP or FlexRay, is is required to have a distributed global clock over all components, which is usually a strong constraint in large-scale systems.

bounds must be true bounds (possibly over-estimated). Moreover, they should be as tight as possible, i.e. not too pessimistic (otherwise, this leads to an over-dimensioning, extra weight and power consumption). Finally, they should be efficient and scalable, to handle modern embedded architectures with hundreds of computers and thousands of flows.

3.1 Main Approaches to Timing Verification

The timing verification approaches can be classified into three main categories: methods from real-time systems (often industrial ones), methods from computer networks performances (like Internet, ATM...) and methods from timed systems (model-checking mainly).

- Model-checking is correct, tight, but not scalable. There exist very-efficient timed model-checkers [5], and even if they are not designed to compute delay, the delay can be found by trying to verify a property like "The message arrives before date D", and to compute the bound by a dichotomic search on D. Such model-checkers have been extended to perform parametric verifications [6] which allows to compute D directly. But even if great advances have been done in algorithms and tools efficiency, model-checking still suffers from the combinatorial explosion of the state space, and cannot be used for large systems.
- Real-time scheduling is a wide research area, with a long history and a lot of diverse results. If most results in real-time scheduling have been used for local scheduling, or distributed scheduling with the network delays as an input of the problem, some studies have been done to handle communications [7], (which associates some well known local methods with a fixed point iteration) or to see one bus as a local scheduling problem [8]. More recently, approaches known as "trajectorial" have been developed [9,10]. Such approaches give good results [10], but the complexity is relatively high: computing the release time of a message created at time t requires solving a non-linear system, and this computation must be repeated for each significant instant, whose number depends on the least common multiple of the period of the flows.
- Network Calculus is a theory to get deterministic upper bounds in networks that has been developed by R. Cruz [11,12], and popularized with two books [1,2]. A nice overview of can be found in [13]. It is mathematically based on the (min,+) dioid and from the modelling point of view, it is an algebra for computing and propagating constraints given in terms of envelops. A flow is represented by its cumulative function $R(t)$ (that is, the amount of data sent by the flow up to time t). A constraint on a flow is expressed by an arrival curve $\alpha(t)$ that gives an upper bound for the amount of data that can be sent during any interval of length t. Flows cross service elements that offer guarantees on the service. A constraint on a service is a service curve $\beta(t)$ that is used to compute the amount of data that can be served during an interval of length t. It is also possible to define in the same way minimal arrival curves and maximum service curves. Then such constraints envelop the processes and the services.

3.2 Why Network Calculus Fits Embedded Systems

Among the other temporal verification techniques, Network Calculus fits well critical embedded systems for several reasons. First of all, it relies on strong mathematical foundations since network calculus is based on the (min,plus) algebra [14] with well-identified mathematical assumptions.

Most often, classical scheduling-based methods are based on the exhibition of a "worst case scenario", which must be translated into an analytical expression, that in turn should be solved to obtain numerical results. This process mainly relies on human reasoning, which could lead to errors, as it happened for the CAN schedulability analysis, considered as solved in [15], and which was found to be slightly flawed and corrected 13 years later in [8]. With its strong mathematical background, Network Calculus is less sensitive to this kind of problem but it is of course sensitive to mathematical aspects, like continuity, limits, etc.

Network Calculus also handles natively multi-hops networks, one of its famous result is the "pay burst only once", which allows seeing multiple hops as a single element. Since this is a generic method, it also handles heterogeneous networks and allows to consider an end-to-end path with, for example, a CAN and an AFDX segment.

A last property is of special interest: Network Calculus allows a lot of sound over-approximations[2]. Such feature enables to reduce the computation time (at the expense of the results accuracy), which could be useful for a coarse-grained design or to evaluate large-scale systems. But, on top of it, it lets the user opt for a simple model, or simple verification algorithms, if the more accurate and complex ones are too hard to verify, qualify or certify (depending on the industrial context).

Of course, these benefits are counterbalanced by the pessimistic approximations made by the theory.

3.3 Network Calculus: An Overview of the State of the Art

In its simplest form, Network Calculus enables to perform the following operations:

- compute the exact output cumulative function or, at least, bounding functions,
- compute output constraints for a flow (like an output arrival curve),
- compute the remaining service curve, that is, the service that is not used by the flows crossing a server,
- compose several servers in tandem,
- give upper bounds on the worst-case delay and backlog (bounds are tight for a single server or a single flow).
- the operations used for this are an adaptation of filtering theory to (min,+): (min,+) convolution and deconvolution, sub-additive closure.

[2] Soundness, in this context, means that the bound computed in the over-approximated model are still bounds, probably pessimistic but always valid.

These possibilities have been extended in several directions. First the notion of service curve can be defined in several manners, depending on the chosen model. To compute remaining service curves, one has to work with strict service curve [16], but to study FIFO scheduling policy, the notion of service curve is enough [17]. Real-Time calculus is mainly based on Network Calculus, but also uses concepts of real-time scheduling theory [18]. For this approach, the elementary operator transforms an arrival curve of a flow and a service curve of a service elements into the output arrival curve and a remaining service curve, and this operator has good compositional properties to study systems with fixed priorities. Finally, there are some adaptations of Network Calculus to Stochastic Network Calculus [2,19]) in order to relax a bit the constraints, but worst-case delay bounds cannot be computed with this theory.

Concerning the results achieved in the Network Calculus field, a lot of studies are now focused on computing performances in networks and composing network elements. Up to our knowledge, the networks that have been extensively studied are: servers in tandem or sink trees [17,20,16]. Some other studies focus on general networks with cyclic dependencies, and exhibit networks where, against intuition, the load is very small in each server but can be unstable (that is, the backlog is not bounded). To get such results, the authors construct ad-hoc scenarios [21]. More compositional methods have been used very recently for both Network Calculus and Real-Time Calculus, giving sufficient conditions to get stability and worst-case delay upper bounds [22,23].

The prominent software tool is Rockwell-Collins ConfGen Tool which is a proprietary tool that uses network calculus to compute traversal times on AFDX. It has been for instance used to validate the delays in AFDX network for Airbus A380. The bounds on end-to-end delays provided by this tool were really larger than what was observed on simulations and experimentations [24]. The PEGASE proposal aims to develop an ambitious successor of this tool providing several improvements and additional features: more realistic bounds, handling of cyclic dependencies, new class of protocols (e.g., wormhole routing) and design assistance. These features must be based on new theoretical investigations.

3.4 Objectives and Novelty of the PEGASE Project

The main idea of the family of Network Calculus techniques is to take into account the traffic characteristics under the form of regulating functions and to model network nodes (switches, filters, cables) as operators acting on those functions.

Thus, Network Calculus operates at the flow level and not at the packet level (unlike the trajectorial approach, for example) and never uses the state space of the system (unlike model checking). These two features of Network Calculus make it perfectly fit to analyse systems whose state spaces are very large or with a combinatorial complexity making them hard to analyse with approaches based on a fine description of their behaviour.

This project has two objectives, one is rather theoretical and aims at improving Network Calculus in terms of control of bounding errors and assertion of its

descriptive power. The other is to demonstrate its usefulness for the design of communicating embedded systems, especially for the aeronautic and the space industry.

Industrial objectives: simplify and tighten. This project focuses on the communication networks of embedded real-time systems, where the worst end-to-end delay must be characterised and taken into account. The industrial objectives are:

- tighten the computed worst end-to-end delay to reduce over-provisioning which leads to extra weight and power consumption,
- simplify the design of such networks (dimensioning, routing) to reduce conception costs,
- provide some analysis tools to facilitate the validation of critical real time embedded data networks transporting different classes of traffic (guaranteed delivery in time, assured delivery, best effort...).

Scientific objectives: pushing the limits of Network Calculus. To achieve these industrial goals, we propose to work on three scientific axes:

1. Eliminate some current over-approximations to tighten the bounds: some preliminary works have shown that the difference between the computed bounds and the real worst case comes from the poor modelling of some characteristics of embedded network elements. For example, the non-preemptive aspects are handled in a pessimistic way. Furthermore, the global analysis is often done by the recursive use of local results, and it has been shown in [16] that it could produce very pessimistic results. New global methods should be developed.
2. Extend the class of systems that can be modelled and analysed: up to now, some systems can not or incompletely be handled with Network Calculus (cyclic dependencies between flows, wormhole routing). So far, such configurations must be avoided if one wants to apply Network Calculus formulas. We aim at removing these restrictions.
3. Provide help in the design (dimensioning, routing): in the current network design process, the worst end-to-end delay is taken into account a posteriori (once the system is designed, worst-case bounds can be computed to validate the design). On the opposite, in classical scheduling theory, there exist heuristics to help the design (priorities, placement). We would like to develop similar methods in the Network Calculus framework to help with dimensioning and routing in embedded networks.

There are of course scientific hard points, the main challenges addressed by PEGASE can be grouped into three main points:

1. Semantics relations: Network Calculus have been extended in various directions for various purposes, but the differences between model hypotheses are not always clear. In particular, they all use notions of arrival curve and

service curve, sometimes without precise definitions of the assumptions or without discussions about the conditions of application. What real system behaviour can be modelled by each definition? Some preliminary results can be found in Section 5.

2. Modelling granularity: one main asset of Network Calculus is the liberty in the modelling granularity. A complex behaviour can always be approximated by a lower (resp. upper) service curve (resp. arrival) curve. Such approximation is conservative, i.e. the results obtained with the coarse model are still valid but often not very tight. But, in general, the precise modelling is too computation consuming. Then, a trade-off between computation cost and tight modelling must be find.
3. Are shared memory flow control protocols (min,plus) systems?: All Network Calculus flavours are based on some (min,plus) theory. But it is not clear whether flow control protocols with shared memory (like wormhole routing used in SpaceWire or some NoC) are such kind of systems.

4 Case-Studies : AFDX, SpaceWire and NoC

To assess the extent to which network calculus can be an effective tool in various industrial contexts, three real applications based on distinct communication protocols are used as case-studies during the project.

4.1 AFDX

The AFDX technology (ARINC 664 standard, part 7, [3]) is an embedded network based on the Ethernet technology, in order to take benefit of a largely deployed technology and reach acceptable costs. Ethernet offers large bandwidth, but suffers from indeterminism (in particular, the well known random back-off algorithm involved in collision). In AFDX, each end-system is connected to a switch with full-duplex links: there is no more collision, and indeterminism comes only from waiting time in switch shared queues. To get guarantees on this shared network, incoming flows must respect some traffic contract. In AFDX, this contract is a *Virtual Link* (VL). Each VL is a static multicast mono-source route in the net (a tree), with a priority level, and three parameters related to bandwidth: a minimal frame size, a maximal frame size, and a *Bandwidth Allocation Gap* (BAG) which is the minimum time between two frame emissions by the source.

A typical AFDX network could have a dozen of switches, hundred of connected equipments, thousand of VL, and a total incoming throughput between 150Mb/s and 200Mb/s. Given an AFDX configuration, it is possible to compute upper bounds on delays and buffer sizes (see for instance [25,26]), however there are still some challenges left:

- more accurate temporal verification that enables to obtain the same guarantees on performance with less embedded switches,
- automatic configuration design.

4.2 SpaceWire

The purpose of the SpaceWire standard [4] is:

- to facilitate the construction of high-performance onboard data-handling systems,
- to help reduce system integration costs,
- to promote compatibility between data-handling equipment and sub-systems,
- to encourage re-use of data handling equipment across several different missions.

The SpaceWire network enhances the communication capabilities, but induces more complexity in the traffic management, so requiring enhancement of classical methods of validation. In addition, high speed network (for command/control data) is a new field of embedded designs for Space. Traffic complexity increases is due to (i) the presence of routers and switch matrices (presence of multiple application data and sources, wormhole routing); (ii) the overheads introduced by multiple layered protocols; (iii) the SpaceWire standard features (various and high data rates, no bus controller, arbitration within routers, non predictable dispersion in delays, links shared by different data flows...). The new challenges for SpaceWire adoption are about:

- Defining adequate network topology (no bottleneck, redundancy),
- Consolidating communication designs and performances (latency delays),
- Sizing local resources (temporary storage).

Thales Alenia Space provides, as case study, a long-term horizon architecture based on a single and common SpaceWire network for the transport of command/control and mission (science, observation) data. It consists of 14 nodes (9 instruments and 5 platform equipments) and 4 routers. The Figure 1 shows an overview of the architecture. It deals with 39 data flows (9 mission flows and 30 control/command flows). Flows are asynchronous and follows a known policy for production / consumption of data.

4.3 Network on Chip

Integration is a multi-level concern in embedded systems. Shared network is now the current solution on embedded chips, either as interconnecting different devices on a single chip (system on chip - SoC) or replacing the bus on multi-core CPU. To compute the worst case execution time (WCET) of code on such hardware, the delay introduced by this internal communication medium must be taken into account. But current trend in chips is to increase performances and reduce predictability. In critical system, it makes no sense to have a chip with high mean performances if the worst case cannot be reasonably bounded. This case study is the most prospective one of the project: several hard issues have been identified. To begin, there are a lot of routing technologies in NoC, and some could be easier to verify than other with network calculus. A first issue is to choose and/or define a manageable NoC. Second main issue is the characterization of the data flow exchanged by the different components in order to help the mapping of the application (Design Space Exploration)

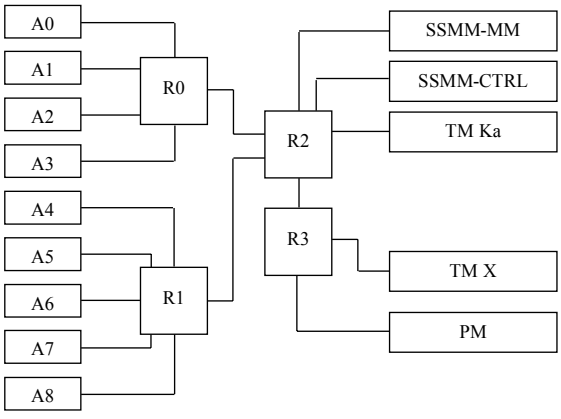


Fig. 1. SpaceWire architecture

5 Some Theoretical Improvements

The first months of the project have been spent to clarify the relationships between the main variants of envelope-based models, like the Real-Time Calculus [18], variable capacity nodes, strict service and minimum service [27]. Some restrictions on strict priority (SP) residual service have also been lifted. Tight bounds have been obtained for the FIFO policy [28,29] with results on the complexity of computing these bounds [29].

5.1 Model Hierarchy

Considering that it exists four main notions of service: RTC, variable capacity nodes, strict service and minimum service, the question of their relationship naturally arises.

When looking deeply at definition, it also appears that the notion of strict service curve can have two interpretations, one weak and one strict one.

It has been shown in [27,30] that these notions forms a strict hierarchy, with equality between notion depending on the kind of curves considered. Some monotony results have been also found for each model, allowing to obtain, in most cases, a canonical version of each service curve.

5.2 Strict Priority Residual Services

When several flows share a network element, one may be interested by the service offered to each one (depending on the service policy and the others flows) which is called the "residual service".

This residual service is often assumed to have a wide-sense increasing curve, restricting the kind of usable curves. This restriction has been lifted in [27], and the links with strict service curve has been made explicit.

5.3 Tight Results under Blind Policy

Network calculus, in general, computes pessimistic bounds. One challenge is to get tight bounds, *i.e.* bounds that could be reached. It is well known that bound are tight for a single server [1, Th 1.4.4]. There also are results for FIFO policy in some specific topologies [31].

[28,29] describe the first algorithm which computes the maximum end-to-end delay for a given flow, as well as the maximum backlog at a server, for *any* feed-forward network topology under blind multiplexing, with concave arrival curves and convex service curves.

5.4 Complexity Problem

The computational complexity of the approach in [28,29] is high, probably supra-exponential. The intrinsic complexity of computing an exact bound have been studied on a specific topology: it is NP-hard [29] (by equivalence with X3C problem, using only two-slopes piecewise linear functions). This complexity suggests that only approximated methods are suited to analyse large systems.

6 Tool Support

The complexity of the targeted systems and of the verification methods imposes the usage of a software tool. But the development of a software tool that implements new mathematical methods and that satisfies the practicability requirements of an industrial context requires preliminary exploratory work and proofs of feasibility. Furthermore, researchers need tools that allow them to evaluate the relevance of the theoretical findings through concrete computations on industrial case-studies. For this reason, a prototype is being developed during the project and it is expected to facilitate a rapid transfer of the outcomes of the project to the industry.

6.1 Requirements on the Tool

The practicability of such a tool in an industrial context depends on several aspects:

- acceptance by the certification authorities,
- contained computation time to obtain the results,
- domain-specific support for creating system descriptions that helps to avoid modeling errors,
- ease of understanding and visualization of the analysis and optimization result.

The usefulness of such a tool in an academic context depends on two main aspects:

- as general as possible models – even if to the detriment of raw performance,
- extendibility that enables exploratory work.

6.2 Design Considerations

Network Calculus uses (min,+)-algebra operations whose complexity is strongly dependent on the considered class of arrival and service curves. The more specific the class of curves are (sufficient for industrial applications), the lower the complexity and the more general the class of curves (needed for research), the higher complexity. In order to solve these and other contradictory requirements between industrial and academic usage, the prototype has been structured into several distinct components (see Figure 2), with several different implementations in some cases.

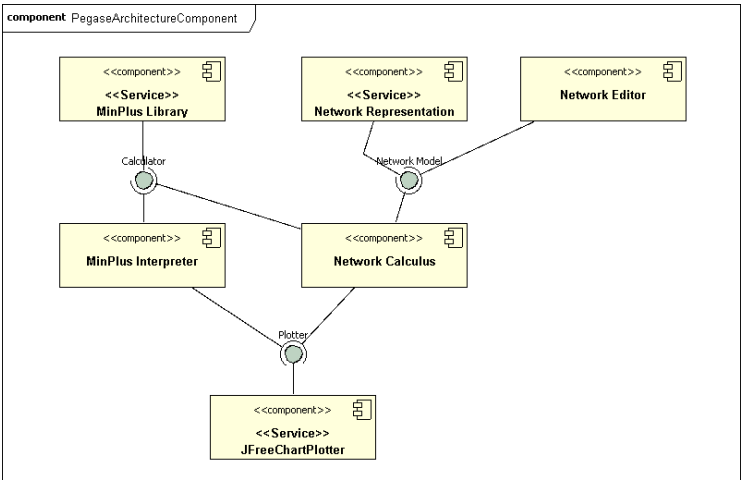


Fig. 2. Components of the prototype (UML notations)

6.3 Implementation

Java has been chosen as programming language for its lower risk for programming errors and because of its link with the Eclipse framework that will be used for the graphical editor. Furthermore continuous integration with frequent releases is performed in order to get rapid feedback from the academic and industrial partners. At the time of writing, the (min,plus) library and the (min,plus) interpreter are fully operational and validated, the network representation (data model) is done and the network calculus core routines are being developed.

6.4 Tool Validation

Given the safety requirements of the application domain, a particular effort is put on the validation of the code:

– Numerous unit tests of the different components of the tools with the mandatory objective of 100% of source code coverage,

– Static analysis of the code with the tool SONAR,
– Extensive automated comparison tests with the Network Calculus tool NC-maude [32].

7 Conclusion

The PEGASE project has been submitted in 2009 to the French call ARPEGE from the ANR (National Research Agency) and it has been selected for funding. The project has started in October 2009 for a duration of 36 months. News from the project, and some of its outcomes, can be found on the project WEB page [33].

The first year of the project already produced some theoretical results (section 5) and the first part of the tool: a (min,+) interpreter (section 6)[3]. These preliminary theoretical results suggest that exact temporal verification techniques will not probably be suited to large scale systems such as avionics ones. The complexity threshold from which approximate techniques are required remains to be more precisely identified. One of the main ongoing objective is to come up with sound approximation techniques, whose accuracy ideally could be chosen by the user.

References

1. Le Boudec, J.Y., Thiran, P.: Network Calculus. LNCS, vol. 2050, p. 3. Springer, Heidelberg (2001), http://lrcwww.epfl.ch/PS_files/NetCal.htm
2. Chang, C.S.: Performance Guarantees in communication networks. In: Telecommunication Networks and Computer Systems. Springer, Heidelberg (2000)
3. AEEC: Arinc 664p7-1 aircraft data network, part 7, avionics full-duplex switched ethernet network. Technical report, Airlines Electronic Engineering Committee (September 2009)
4. ECSS: Spacewire – links, nodes, routers and networks. Technical Report ECSS-E-ST-50-12C, European cooperation for space standardization (ECSS), ESA-ESTEC, Requirements & standards division, Noordwijk, The Netherland (July 31, 2008)
5. Bengtsson, J., Larsen, K.G., Larsson, F., Pettersson, P., Yi, W.: Uppaal - a tool suite for automatic verification of real-time systems. In: Alur, R., Sontag, E.D., Henzinger, T.A. (eds.) HS 1995. LNCS, vol. 1066, pp. 232–243. Springer, Heidelberg (1996)
6. Hune, T., Romijn, J., Va, F.: Linear parametric model checking of timed automata. Journal of Logic and Algebraic Programming (2001)
7. Tindell, K., Clark, J.: Holistic schedulability analysis for distributed hard real-time systems. Microprocess. Microprogram. 40(2-3), 117–134 (1994)
8. Davis, R.I., Burns, A., Bril, R.J., Lukkien, J.J.: Controller area network (CAN) schedulability analysis: Refuted, revisited and revised. Real-Time Systems 35(3), 239–272 (2007)

[3] This interpreter is available free of charge for non-commercial use and can be downloaded at [34].

9. Migge, J.: L'ordonnancement sous contraintes temps réel: un modele á base de trajectoires – Real-time scheduling: a trajectory based model. PhD thesis, Univesity of Nice (1999)
10. Martin, S., Minet, P., George, L.: The Trajectory Approach for the End-to-End Response Times with Non-preemptive FP/EDF*. In: Dosch, W., Lee, R.Y., Wu, C. (eds.) SERA 2004. LNCS, vol. 3647, pp. 229–247. Springer, Heidelberg (2006)
11. Cruz, R.L.: A calculus for network delay, part I: Network elements in isolation. IEEE Transactions on Information Theory 37(1), 114–131 (1991)
12. Cruz, R.L.: A calculus for network delay, part II: Network analysis. IEEE Transactions on Information Theory 37(1), 132–141 (1991)
13. Fidler, M.: A survey of deterministic and stochastic service curve models in the network calculus. IEEE Communications Surveys & Tutorials 12(1) (2010)
14. Baccelli, F., Cohen, G., Olsder, G.J., Quadrat, J.-P.: Synchronization and Linearity, An algebra for discrete event systems. John Wiley and Son, Chichester (1992) ISBN: 978-0471936091, http://cermics.enpc.fr/~cohen-g//SED/book-online.html
15. Tindell, K.W., Burns, A.: Guaranteeing message latencies on controller area network (CAN). In: Proceedings of 1st International CAN Conference, pp. 1–11 (1994)
16. Schmitt, J., Zdarsky, F., Fidler, M.: Delay bounds under arbitrary multiplexing: When network calculus leaves you in the lurch. In: Proc. of the 27th IEEE International Conference on Computer Communications, INFOCOM 2008 (2008)
17. Lenzini, L., Mingozzi, E., Stea, G.: Delay bounds for FIFO aggegates: a case study. Computer Communications 28, 287–299 (2004)
18. Thiele, L., Chakraborty, S., Naedele, M.: Real-time calculus for scheduling hard real-time systems. In: Proceedings of ISCAS 2000 (2000)
19. Jiang, Y., Liu, Y.: Stochastic Network Calculus. In: Computer Communication and Networks. Springer, Heidelberg (2009)
20. Schmitt, J.B., Zdarsky, F.A.: The DISCO network calculator - a toolbox for worst case analysis. In: Proceedings of the First International Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS 2006), Pisa, Italy. ACM, New York (November 2006)
21. Andrews, M.: Instability of FIFO in session-oriented networks. In: Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms (SODA 2000), pp. 440–447. SIAM, Philadelphia (2000)
22. Rizzo, G., Le Boudec, J.Y.: Stability and delay bounds in heterogeneous networks of aggregate schedulers. In: Proc. of the 27th IEEE Conference on Computer Communications (INFOCOM 2008), April 13-18, pp. 1490–1498 (2008)
23. Jonsson, B., Perathoner, S., Thiele, L., Yi, W.: Cyclic dependencies in modular performance analysis. In: Proc. of the ACM International Conference on Embedded Software, EMSOFT (2008)
24. Charara, H., Fraboul, C.: Modelling and simulation of an avionics full duplex switched ethernet. In: Advanced Industrial Conference on Telecommunications/Service Assurance with Partial and Intermittent Resources Conference/E-Learning on Telecommunications Workshop (AICT/SAPIR/ELETE 2005), pp. 207–212. IEEE Computer Society, Los Alamitos (2005)
25. Frances, F., Fraboul, C., Grieu, J.: Using network calculus to optimize AFDX network. In: Proceeding of the 3thd European congress on Embedded Real Time Software (ERTS 2006), Toulouse (January 2006)

26. Boyer, M., Fraboul, C.: Tightening end to end delay upper bound for AFDX network with rate latency FCFS servers using network calculus. In: Proc. of the 7th IEEE Int. Workshop on Factory Communication Systems Communication in Automation (WFCS 2008), IEEE Industrial Electrony Society, May 21-23, pp. 11–20 (2008)
27. Bouillard, A., Jouhet, L., Thierry, E.: Service curves in Network Calculus: dos and don'ts. Research Report RR-7094, INRIA (2009)
28. Bouillard, A., Jouhet, L., Thierry, E.: Tight performance bounds in the worst-case analysis of feed-forward networks. In: Proc. of the 19th IEEE International Conference on Computer Communications (IEEE INFOCOM 2010), pp. 1–9 (14-19, 2010)
29. Bouillard, A., Jouhet, L., Thierry, E.: Tight performance bounds in the worst-case analysis of feed-forward networks. Research Report RR-7012, INRIA (2009)
30. Bouillard, A., Jouhet, L., Thierry, E.: Comparison of different classes of service curves in network calculus. In: Proc. of the 10th International Workshop on Discrete Event Systems (WODES 2010), Technische Universität Berlin, August 30 - September 1 (2010)
31. Lenzini, L., Martorini, L., Mingozzi, E., Stea, G.: Tight end-to-end per-flow delay bounds in FIFO multiplexing sink-tree network. Performance Evaluations 63, 956–987 (2005)
32. Boyer, M.: NC-maude: maude for computation of worst bounds on real-time (embedded) networks. Technical Report 1/16417, ONERA (2010)
33. Boyer, M.: PEGASE home page (2010), http://sites.onera.fr/pegase
34. RealTime-at-Work (RTaW): Downloadable software, http://www.realtimeatwork.com/?page_id=1217

NC-Maude: A Rewriting Tool to Play with Network Calculus

Marc Boyer

ONERA – Toulouse, France
Marc.Boyer@onera.fr

Abstract. Embedded real-time systems are more and more distributed communicating systems. Then, to ensure correctness of application, respect of task dead-line must be ensured, but communication delays must also be bounded. Network calculus is a theory designed to compute such bounds (it have been successfully applied on A380 AFDX backbone). In order to disseminate, and to experiment new results, a tool is needed. Unlike other tools, its purposes are to be open, to allow the user to see the class of function manipulated (sub-additive, star-shaped, concave), the theorems used to get results, etc. To get a code as close as possible to the mathematical context, we chose to use a rewriting language, Maude.

1 Introduction

Networks introduce delays in communications between computers. When these computers run critical real-time applications, like in avionic systems, the delays introduced by the network must be bounded.

The AFDX is the technology chosen by Airbus for its backbone. AFDX is an Ethernet based technology. AFDX have been chosen because it offers a large bandwidth (100Mb/s), allow connections of hundreds of computers, and is a mature technology. It uses full duplex links to avoid collisions and the indeterministic random backoff retransmission algorithm. Then, the only indeterminism comes from the delays in shared switch queues.

Among other theories, network calculus (NC) was chosen to certify the AFDX backbone in Airbus A380 [Gri04]. Network calculus allows to compute upper bounds on network delays, assuming that one have constraints on the incoming flows (some traffic contract, expressed as an *arrival curve*), and guaranties on the service offered by the network elements (expressed as a *service curve*).

From the first works on network calculus [Cru91a, Cru91b] to recent works [BT07, LMS04], with two reference books [LBT01, Cha00], and several extensions [TCN00], network calculus is an active research area. Network calculus have a solid mathematical background: it is based on the (min,plus) algebra [BCGQ92].

Network calculus have been chosen for avionic embedded systems among other theories (based on scheduling theories) because it allows to handle very large systems (thousands of data flows), at the expense of pessimistic approximation on computed delay bounds.

T. Margaria and B. Steffen (Eds.): ISoLA 2010, Part I, LNCS 6415, pp. 137–151, 2010.

The need of an open tool arises because of several reasons:

1. a *tool* is needed because the size of the targeted systems excludes all hand-made results,
2. such a tool must be *open* because the domain is very active, to allow each researcher to add its results to a "state of the art" code,
3. moreover, teaching and disseminating network calculus is facilitated by the existence of an open tool, which allow to play with different elements of the theory, not a complete configuration.

Of course, there are already some tools, but they either are incomplete, or dedicated to specific extensions, or linked to no free tools.

To begin, Section 2 presents network calculus. Section 3, refines this need of developing another tool. Section 4 presents the reasons of the choice of rewriting. Section 5 is an overview of NC-maude, the Maude code dedicated to computations in network calculus. Section 6 concludes.

Appendix A describes a technical detail about tools DISCO and RTC.

2 Network Calculus

Notations $\mathbb{R}$ is the set of real numbers, $\wedge$ denotes the minimum operator, and $[x]^+ \stackrel{\text{def}}{=} max(0, x)$.

Here is a (very short) introduction to network calculus. The reader should refer to [Cha00, LBT01] for a complete presentation.

Network calculus is a theory to get deterministic upper bounds in networks. It is mathematically based on the $(\wedge, +)$ dioid ($\wedge$ denotes the minimum operator).

Network calculus mainly handles non decreasing functions, null before 0: $\mathcal{F}$. They are, among others, four common parametrised curves, δ_d, λ_R, $\beta_{R,T}$, $\gamma_{r,b}$, defined by: $\delta_d(t) = 0$ if $t \leq d$, ∞ otherwise, $\lambda_R(t) = Rt$, $\beta_{R,T}(t) = R[t-T]^+$, and $\gamma_{r,b}(t) = rt + b$ if $t > 0$, 0 otherwise.

Three basic operators on $\mathcal{F}$ are of interest, convolution $*$, deconvolution $\oslash$ and the sub-additive closure f^*.

$$\mathcal{F} = \{f : \mathbb{R} \to \mathbb{R} \,|\, x < y \implies f(x) \leq f(y), x < 0 \implies f(x) = 0\} \quad (1)$$

$$(f * g)(t) = \inf_{0 \leq u \leq t} (f(t-u) + g(u)) \quad (2)$$

$$(f \oslash g)(t) = \sup_{0 \leq u} (f(t+u) - g(u)) \quad (3)$$

$$f^* = \delta_0 \wedge f \wedge (f * f) \wedge (f * f * f) \wedge \cdots \quad (4)$$

A flow is represented by its cumulative function $R \in \mathcal{F}$, where $R(t)$ is the total number a bits send by this flow up to time t. A flow R is said to have a function $\alpha \in \mathcal{F}$ as *arrival curve* iff $\forall t, s \geq 0 : R(t+s) - R(t) \leq \alpha(s)$. It means that, from any instant t, the flow R will produce at most $\alpha(s)$ new data in s time units. An equivalent condition, expressed in the $(\wedge, +)$ dioid is $R \leq R * \alpha$. If α is an arrival curve for R, also is α^*. A server has a *service curve* β iff for all arrival flow, the

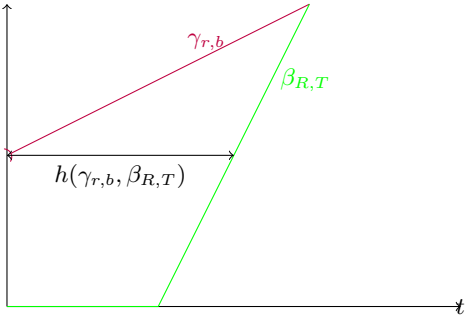


Fig. 1. Common curves and delay

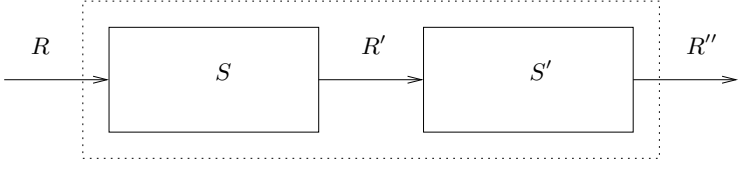


Fig. 2. A flow going through two network elements in sequence

relation $R' \geq R * \beta$ holds between the input flow R and the output flow R'. In this case, $\alpha' = \alpha \oslash \beta$ is an arrival curve for R'. The delay experimented by the flow R can be bounded by the maximal horizontal difference between curves α and β, formally defined by $h(\alpha, \beta)$ (a graphical interpretation of h can be found in Figure 1).

$$h(\alpha, \beta) = \sup_{s \geq 0} \left(\inf \left\{ \tau \geq 0 \,\middle|\, \alpha(s) \leq \beta(s + \tau) \right\} \right)$$

These first results allow to handle linear topologies, like the one of Figure 2. Given the arrival curve α of flow R, and the services curves β, β' of network elements S, S', we are able to compute a bound on the delay in S: $h(\alpha, \beta)$, and another for the delay in S': $h((\alpha \oslash \beta)^*, \beta')$.

In case of more realistic topology, when a network element is shared by different flows, with some service policy (FIFO, static priority, etc.), it also exists results to compute bounds on each flow. For example, with the non preemptive static priority policy, it is known[1] that the lower priority flow has the residual service $\beta_L = [\beta - \alpha_H]^+$, if the network element has service curve β and the high priority flow has arrival curve α_H.

Considering that the flow R_1 of Figure 3 has the low priority, the delay in S can be bounded by $h(\alpha_1, [\beta - \alpha_2]^+)$ and the delay in S' can be bounded by $h((\alpha_1 \oslash [\beta - \alpha_2]^+)^*, [\beta' - \alpha_3]^+)$.

[1] To be precise, there is a restriction on the flavor of service, which must be *strict*. See [LBT01, Def 1.3.2, Cor. 6.2.1,] for details.

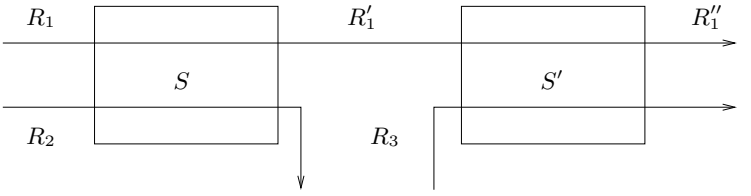


Fig. 3. A topology with shared network elements

This little example shows the three main features of a network calculus tool: encoding the network (topology and characteristics), applying network calculus results, computing $(\wedge, +)$ operations on curves.

Lets us consider the last one, $(\wedge, +)$ operations: these operators are mathematically defined, a lot of properties have been shown (distributivity, isotonicity...), but the effective computation is defined only on some specific classes (mainly piecewise linear functions, concave, convex, pseudo-periodic [BT07]), and the computation cost of each operator depends on the considered class. For example, if f, f' are concave functions, $f * f' = f \wedge f'$, $f^* = f$. In the general case of piecewise linear pseudo-periodic, such computations are polynomials.

The second one, applying network calculus results, is the main objective of the tool: the system must be open to allow users to apply different theorems on configurations, to see the effects of different modellings, and to add new results.

Last feature, encoding the network, is just data representation.

3 WhAT: Why Another Tool?

They are several tools already developed for network calculus. As presented in introduction, the size of the targeted systems (thousands of flows) prevents from any hand-made system analysis, and any researcher in the area must have some running code to experiment its results. But, each time, the same question arises: going from an existing tool or developing a new one.

One the one hand, developing a tool is a good way to get experience in a domain, but it could be very time consuming. On the other hand, extending an existing academic source could be harder than rewriting it from scratch[2]...

But here, they are also others reasons.

3.1 DISCO

The DISCO network calculator [GZMS08, SZ06, DIS] is developed by the DISCO research group of technical university of Kaiserslautern (Germany). As presented in the web page, it is a network calculus library developed in Java, and "the core of this library is the class "Curve" which represents piecewise-linear curves and provides min-plus-algebraic operations on these."

[2] Especially when the code uses `break` and `continue` instructions.

DISCO is as open as a Java code could be: it is mainly an API, which allow to create curves and topologies, and also to compute delays, using several analysis methods.

The kind of curves manipulated is wide-sense increasing piece-wise linear functions, with a finite set of segments. This is a smaller class than the one described in [BT07] and implemented in [COI].

DISCO is made of around 5000 lines of Java code and 500 lines of comments.

3.2 COINC

COINC is the implementation of the algorithms presented in [BT07]. The aim of [BT07] was to find a manipulable (i.e. with algorithms) class of functions closed under the operators used in network calculus (sum, difference, minimum, maximum, convolution, deconvolution, and sub-additive closure). The piece-wise linear functions, with a finite set of segments, seems a good candidate (this is the choice of [GZMS08]). But the sub-additive closure goes out of this class, and one need to consider the set of piece-wise finally pseudo-periodic functions (with rational values) to keep a class closed by all operators.

In [BT07], the algorithms for all operators in this class are given, and their implementation is the aim of [COI]. COINC activity have been stuck a few years as an interpreter where sub-additive closure was not implemented. During year 2009, the implementation have been completed and the C++ code can be called from Scilab [Sci].

COINC is made of 4000 lines of C++ with about 1000 lines of comments, mostly in french.

3.3 CyNC

The Cyclic Network Calculus tool (CyNC, [Sch]) have been developed to illustrate the ability of network calculus to handle cyclic dependencies [SJDL05, SNLJ06].

The tool is a Matlab/Simulink library, which allow to graphically describe a system, and analyse it.

The project seems to be frozen (the last update on [Sch] is from July 2005).

CyNC is made of 800 lines of Matlab code, with 26 lines of comments, without any explanation on the data model, which makes its reuse very hard.

3.4 RTC

The Real-Time Calculus Toolbox [WT06] is presented as a Matlab library, but it is more than that: it is a Java library with an API, and a Matlab interface.

The kind of curve manipulated by RTC is the same than COINC: piece-wise affine functions finally pseudo-periodic. Most operators are implemented, except sub-additive closure.

The theory implemented in RTC is not the basic network calculus, but the Real-Time Calculus, an extension of the basic theory [TCN00], and a specific solution for cyclic dependencies MPA [TS09].

RTC is made of 4000 lines of Matlab code, and 3500 lines of comments. The Java code is not distributed, but the API is distributed under html format, generated by javadoc, and presents 22 classes.

3.5 DEBORAH

DEBORAH is devoted to the computation of bounds in networks with FIFO policy, rate-latency services and token-bucket arrival curves [BLMS10]. It implements the results of [LMS04, LMMS05, LMS07]. It computes upper bound on delay. This implies solving some piecewise-linear programming problems[3]. It also computes lower bounds on the worst delay, considering specific behaviour.

DEBORAH is made of 6000 lines of C++, and about 1500 comment lines. It is distributed under the GPL licence.

3.6 PEGASE

The french national project PEGASE is devoted to aerospace communication systems, including new results in network calculus and industrial transfer [PEG10, BNOT10]. In this context, the company RealTime-at-Work develops a network calculus tool [Rea]. It is decomposed in several blocs: one min-plus computing library, a network calculus core, and some professional features like a user friendly GUI, network editor, etc [BNOT10, § 6].

The min-plus part is a Java implementation of the results of [BT07], and is free for academic use. This module has the same objectives as COINC (see section 3.2) but has been re-developed for several reasons, including intellectual properties, implementation language, code maintenance, code validation, etc. To increase its confidence, it imports (and passes) the unitary tests of NC-maude.

3.7 NC-Maude Objectives

Once we have presented the other tools, what are the design challenges of a new one?

The aim is to be more "open" for beginners: having a syntax close to the mathematics used in the papers, having an interpreter to allow to interactively manipulate the different elements, and manipulating $(\wedge, +)$ and network calculus object in the same environment.

The goal is not to have a black-box, with user-friendly GUI, taking a network topology as input and giving some bounds as output. Such tool is, of course, necessary to an industrial use, but is not NC-maude goal.

Let make some comparisons with other tools.

Compared to COINC, the goal was to consider a smaller class of functions, with simpler (and so more readable) implementation, but also to implement the network calculus results, not only the $(\wedge, +)$ algebra.

[3] The complexity is theoretically exponential, but some heuristics keep the computation time reasonable on real example.

All of these tools handle coarsely the types of functions: they all works with piecewise linear functions, but information such as concavity are not in the type system, and the code is full of test like `if (!isConvexe(...))`. And such type information are useful in this context: for example, it is known that the convolution of two concave functions, both null at zero, is equal to the minimum of such functions (which is easier to compute).

The NC-maude code handles very finely variety types of functions, wide-sense increasing, null at 0, star-shaped, concave, convexe, etc.

DISCO implements network calculus results, but does not have any interpreter or interface. One must write Java code to define functions and net topologies.

The DISCO and RTC tools made some approximations with the mathematical background, on continuity for example, which does not affect the quality of the results, but could be disturbing for beginners. In the RTC tutorial is written the following "The various functions of the RTC Toolbox will interpret a discontinuity correctly as either left- or right-continuous depending on the context of the curve.". An example of such approximation is presented in appendix A. NC-maude tries to be strictly conform to the theory.

At least, the relationships between RTC and NC are, up to now, unclear, and our goal was dissemination of NC.

Its should also be free to help dissemination: any Matlab toolbox is linked to Matlab, which is an affordable tool for academics, but not really a free tool.

A last difference with other tools: NC-maude uses rationals, not floating point numbers. It takes more computation time, but gives tractability to the results.

The author of NC-maude is also involved in the PEGASE project, where the PEGASE tool is currently developed. It implies some common points between both project (the min-plus module is written in Java, but the network-calculus module should use some rewriting based language). But NC-maude is designed to be an academic open prototype, since PEGASE is designed to be an industrial tool, which can be used by networks engineers without any deep knowledge of NC theory.

4 Why Rewriting?

Note: in this section, we are using the term "rule" as a generic term that describes a rewriting pattern on terms, without distinction between equations, rule, conditional or not, as made in some tools.

Once the decision to write an open extensible tool was made, the choice of a language had to be done.

We chose to use rewriting-based language for the following reasons.

1. The *syntax* of a rewriting based tool expresses very naturally *mathematical equations*, and since networks calculus have a very formal mathematical part, a rewriting based code have several benefits.

   Readability. The code is very closed to the problem itself, sorts, rewriting rules, are very closed to the mathematical definitions and theorems,

Tractability. Each non-trivial rule can be labelled by the reference of the theorem is implements. The link between specification and the code is really easy to do.

Confidence. Since the kernel of a rewriting tool is just the application of individual rules, the confidence into the results comes from the tractability on each rule. Moreover, if necessary, the rewriting trace can be used as a proof, since it is often easier to certify a specific proof than a generic tool.

2. Rewriting tools come with efficient matching algorithms, dedicated to mathematical based properties (associativity, commutativity, etc.), and exonerate the programmer to recode it.

Of course, when using a specific language, some questions arise about the difficulty to add its own new algorithms or the efficiency of the programs. Sections 5.3 discusses this point.

5 NC-Maude

The aim of NC-maude [Boy] is to be an open, extensible tool, designed for dissemination of NC theory. To achieve this goal of dissemination, we want the interactions from the user with the tool to be very close to the theory. Like other tools, the user must be able to describe a topology and to ask for bounds on flows. But, unlike others, it uses a syntax really closed to the theoretical papers, it allows the user to ask to the tool if a given function is known as star-shaped, to know which theorem is used at some point, etc. NC-maude uses rational computation: it costs more CPU than simple floating point numbers, but tractability is easier.

Graphical output is done by generating gnuplot command which must be copied/pasted by the user into gnuplot.

5.1 An Example of Interaction

An example could gives a quicker overview than general description. Then, here are some examples of interactions.

Keep in mind than NC-maude is designed to be a tool to support discovering of NC. In this illustrative section, we are going to use some domain-specific objects, like common functions $\gamma_{r,b}$, $\beta_{R,T}$, or operators (horizontal deviation h), presented in Section 2.

The Figure 4 shows an interaction example, on the $(\wedge, +)$ domain.

First request, line 8, is the sum of terms $\gamma_{5,2} + \gamma_{3,5}$. The tool computes the result, and gives the type of the result. We can wonder if the result is concave. This is done by using the built-in operator `::` of Maude (line 11). And the interpreter answers it is. A last request in the $(\wedge, +)$ world is the computation of the horizontal deviation between two curves: $h(\gamma_{4,5}, \beta_{10,2})$ (line 14). Notice that the syntax of the request is really closed to the mathematical syntax.

```
boyer-laptop% maude startNC.maude
                     \||||||||||||||||||/
                   --- Welcome to Maude ---
                     /||||||||||||||||||\
            Maude 2.4 built: Nov  6 2008 17:14:50
            Copyright 1997-2008 SRI International
                   Fri Aug 21 13:29:56 2009
Maude>red (Gamma 5 2 ) + (Gamma 3 5 ) .
reduce in START-NC : Gamma 3 5 + Gamma 5 2 .
result Gamma: Gamma 8 7 .
Maude> red Gamma 8 7 :: Concave .
reduce in START-NC : Gamma 8 7 :: Concave .
result Bool: true
Maude> red h(Gamma 5 3, Beta 9 2 ) .
reduce in START-NC : h(Gamma 5 3,Beta 9 2) .
result PosRat: 7/3
Maude> parse flow("R",Gamma 5 3) .
Flow: flow("R",Gamma 5 3)
Maude> parse  netElem("S",Beta 9 2) .
NetworkElement: netElem("S",Beta 9 2)
Maude> red delay( flow( "R" , Gamma 5 3 ) ,  netElem( "S" , Beta 9 2 )  ) .
reduce in START-NC : delay(flow("R",Gamma 5 3),netElem("S",Beta 9 2)) .
result PosRat: 7/3
Maude> red flow( "R" , Gamma 5 3 ) |  netElem( "S" , Beta 9 2 ) .
reduce in START-NC : flow("R",Gamma 5 3) | netElem("S",Beta 9 2) .
result Flow: flow("R|S",Gamma 5 13 )
```

Fig. 4. NC-maude interaction illustration

```
mod SIMPLE-EX is
  protecting START-NC .
  *** Constant declaration
  ops r r' : -> Flow .
  ops s s' : -> NetworkElement .
  *** Constant definition
  eq r = flow( "R", Gamma 5 4 ) .
  eq r' = r | s .
  eq s = netElem( "S" , Beta 10 2 ) .
  eq s' = netElem( "S'" , Beta 20 1 ).
endm
red r' .
red r | s | s' .
red delay( r , s ) .
red delay( r' , s' ) .
red delay( r , s ; s' ) .
```

Fig. 5. NC-maude module describing the simple architecture of Figure 2

The user could also notice that the delay result is given as a rational, not a floating point number.

To describe a topology, some domain-specific operators have been defined. A flow can be described by its name and it arrival curve (line 17), and a server can

be described by also its name and its service curve (line 19). We can then ask the tool to compute the delay experimented by the flow in the server (line 21), and also the output flow (denoted with the pipe operator, by analogy with Unix shell syntax – line 24).

To describe a complete topology, it is easier to define constants (operators without argument in Maude) in a module. A simple example in given in Figure 5. It describes the architecture presented in Figure 2 where a flow R goes in sequence through two servers S and S'.

5.2 NC-Maude Code Description

The NC-maude tool is decomposed into three main parts (could be named *packages* in another language).

The first one is the representation of the $(\wedge, +)$ algebra. It defines some sorts on the set $\mathcal{F}$ (see section 2) of functions (denoted naturally `F` in the Maude code), the common operators (sum, min, max...), some of they basic properties (distributivity), and some domain-specific operators (convolution and deconvolution). It also defines some common functions and they related equations.

This is done in a syntax very closed to the domain, as illustrated in the following lines, extracted from the code.

```
1   sort F .
    op _ + _ : F F -> F [assoc com] .
    op _ /\ _ : F F -> F [assoc com] .
    vars f g h : F .
5   eq [DistAddMin] : f + ( g /\ h ) = (f + g ) /\ ( f + h )  .
    op Gamma _ _ : Rat Rat -> F .
    ceq [MinGammaGamma] :
          (Gamma r b) /\ (Gamma r' b') = Gamma r b
          if r <= r' /\ b <= b'
10  eq [SumGammaGamma] :
          (Gamma r b) + (Gamma r' b') = Gamma (r + r') (b + b') .
```

One specific sub-sort of function in $\mathcal{F}$ is the set of concave piece-wise linear (CPL) functions . Such a function can be represented as the minimum of a set of γ functions. This interpretation is sufficient to manipulate such functions, based on the individual properties of minimum and other operators (for example, the sum of two CPL functions can be implemented using distributivity of sum over minimum and the sum of two γ functions). But even is such interpretation is true, it could be inefficient. One representation of such functions as a sorted set of segments allows for example to implement sum with a linear complexity (cf Figure 6). Then, the second part of the system is dedicated to an efficient implementation of the basic operations on the CPL sort.

The third part of the tool is dedicated to the network calculus itself: representations of the flows, of the servers, and equations encoding the main theorems of the theory. For example, the code to compute an output flow from input flow

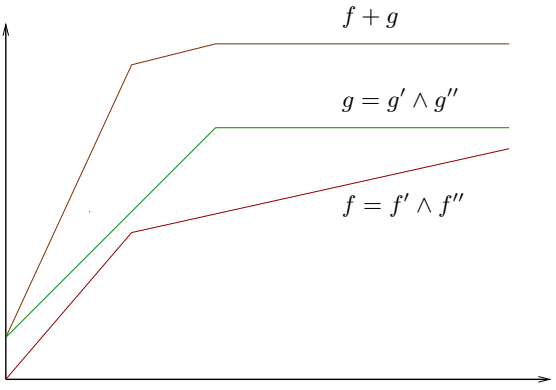


Fig. 6. Sum of CPL functions

and network element or to compute a bound on delay of a flow in a network element look like the following.

```
var aflow : Flow .
var Name : String .
var SC : F .
eq [arrival-deconv-th-1.4.3] :
   aflow | netElem( Name , SC )
   =
   flow( name(aflow) + "|" + Name ,
         arrivalCurve(aflow) deconv SC ) .
eq delay( aflow,  netElem( Name, SC ) )
   =
   h( arrivalCurve(aflow), SC) .
```

At current time, NC-maude has 33 sorts, about 800 equations (with 250 conditional equations), and about 650 unitary tests. It contains 4000 lines of code, and 1400 comments lines (without unitary tests).

The implementation was a background task over more than tree years, and it is hard to determine the coding effort. It should be about 6-9 months...

5.3 Extending NC-Maude

NC-maude has been designed to be open, to allow any one to add and test its new algorithms. But what is the related difficulty?

They are mainly two origins: the first is the use of an unknown language, Maude, the second is the architecture of the program NC-maude itself.

On the use of Maude, our feedback can be found in [Boy10]. In a few words, Maude can be seen as a pure functional language, very well designed for local operations on terms.

The architecture of the code, as presented in previous section, is based on the mathematical framework of network calculus, as presented in [LBT01]. Its had been designed to be modular and extensible, for people already aware of the NC theory.

One can also wonder about the performances. NC-maude has not been designed for performances: for example, its handle about 10 sorts of functions, when in most common cases, only one is sufficient (the set of concave piecewise linear function, CPL). As well, it was designed to use rational numbers, with infinite precision (even if a floating point version is under development).

Nevertheless, the performances are acceptable for a prototype: for a configuration with 7000 flows and more than 250 queues, on a laptop with a 1,4GHz processor, its takes on average 1.5s per queue to compute the delay bound of the queue, on the floating point version, and 10,5s on the rational version.

5.4 Licence

NC-maude is distributed under the CeCILL licence, a GNU-GPL compatible licence. This new licence have be developed because the use of licenses created in the US, such as the GNU General Public License raises some legal issues in France. To provide a better legal safety while keeping the spirit of these licenses, three French public research organisations, the CEA, the CNRS and INRIA have launched a projet to write Free Software licenses conforming to French law: the CeCILL licence. CeCILL is the first license defining the principles of use and dissemination of Free Software in conformance with French law, following the principles of the GNU GPL.

6 Conclusion

We have developed a new tool to disseminate network calculus. Its main characteristics are: openness, rewriting language, and direct map of mathematical background.

NC-maude is written in Maude, a rewriting language. The $(\wedge, +)$ part is implemented mainly as a direct translation from the mathematical domain[4], and the properties of the $(\wedge, +)$ algebra are rewriting rules, used to reduce term *i.e.* compute results (sum, convolution, deconvolution, etc). The encoding on networks (flows, servers, topology) is also done in an algebraic way. The theorems of network calculus are also rewriting rules. Computing delay bound on a flows through a network is the recursive application of local reduction rules, and topology traversal.

Any network calculus beginner can make requests in a syntax very close of the one used in theoretical papers, and have access to all step of reduction, if needed.

It is internally used for years at ONERA, and can now be adopted by new users.

[4] For performance reasons, the specific class of piecewise linear concave functions is implemented in a more algorithmic way.

References

[BCGQ92] Baccelli, F., Cohen, G., Olsder, G.J., Quadrat, J.-P.: Synchronization and Linearity, An algebra for discrete event systems. John Wiley and Son, Chichester (1992) ISBN: 978-0471936091, http://cermics.enpc.fr/~cohen-g//SED/book-online.html

[BLMS10] Bisti, L., Lenzini, L., Mingozzi, E., Stea, G.: DEBORAH: a tool for worst-case analysis of FIFO tandems. In: Margaria, T., Steffen, B. (eds.) ISoLA 2010. LNCS, vol. 6415, pp. 210–226. Springer, Heidelberg (2010)

[BNOT10] Boyer, M., Navet, N., Olive, X., Thierry, E.: The PEGASE project: precise and scalable temporal analysis for aerospace communication systems with network calculus. In: Margaria, T., Steffen, B. (eds.) ISoLA 2004. LNCS, vol. 6415, pp. 180–194. Springer, Heidelberg (2010)

[Boy] Boyer, M.: NC-maude home page, http://www.onera.fr/staff/marc-boyer/tools.php

[Boy10] Boyer, M.: NC-maude: maude for computation of worst bounds on real-time (embedded) networks. Technical Report 1/16417, ONERA (2010)

[BT07] Bouillard, A., Thierry, É.: An algorithmic toolbox for network calculus. Discrete Event Dynamic Systems 17(4) (October 2007), http://www.springerlink.com/content/876x51r6647r8g68/

[Cha00] Chang, C.-S.: Performance Guarantees in communication networks. In: Telecommunication Networks and Computer Systems. Springer, Heidelberg (2000)

[COI] COINC home page, http://www.istia.univ-angers.fr/~lagrange/software.php

[Cru91a] Cruz, R.L.: A calculus for network delay, part I: Network elements in isolation. IEEE Transactions on Information Theory 37(1), 114–131 (1991)

[Cru91b] Cruz, R.L.: A calculus for network delay, part II: Network analysis. IEEE Transactions on Information Theory 37(1), 132–141 (1991)

[DIS] The DISCO network calculator home page, http://disco.informatik.uni-kl.de/content/Network_Calculator

[Gri04] Grieu, J.: Analyse et évaluation de techniques de commutation Ethernet pour l'interconnexion des systèmes avioniques. PhD thesis, Institut National Polytechnique de Toulouse (INPT), Toulouse (Juin 2004)

[GZMS08] Gollan, N., Zdarsky, F.A., Martinovic, I., Schmitt, J.B.: The DISCO Network Calculator. In: 14th GI/ITG Conference on Measurement, Modeling, and Evaluation of Computer and Communication Systems (MMB 2008), Dortmund, Germany, GI/ITG (March 2008)

[LBT01] Le Boudec, J.-Y., Thiran, P.: Network Calculus. LNCS, vol. 2050. Springer, Heidelberg (2001), http://lrcwww.epfl.ch/PS_files/NetCal.htm

[LMMS05] Lenzini, L., Martorini, L., Mingozzi, E., Stea, G.: Tight end-to-end per-flow delay bounds in fifo multiplexing sink-tree network. Performance Evaluations 63, 956–987 (2005)

[LMS04] Lenzini, L., Mingozzi, E., Stea, G.: Delay bounds for FIFO aggegates: a case study. Computer Communications 28, 287–299 (2004)

[LMS07] Lenzini, L., Mingozzi, E., Stea, G.: End-to-end delay bounds in fifo-multiplexing tandems. In: Glynn, P. (ed.) Proc. of the 2nd International Conference on Performance Evaluation Methodologies and Tools (ValueTool 2007), Nantes, France, October 23-25, ICST (2007)

[PEG10] Pegase home page (2010), http://sites.onera.fr/pegase
[Rea] RealTime-at-Work. Realtime-at-work home page, http://www.realtimeatwork.com
[Sch] Schiøler, H.: CyNC home page, http://www.control.auc.dk/~henrik/CyNC/
[Sci] Scilab consortium. Scilab hme page, http://www.scilab.org
[SJDL05] Schiøler, H., Jessen, J.J., Dalsgaard, J., Larsen, K.G.: Network calculus for real time analysis of embedded systems with cyclic task dependencies. In: Proceedings of the 20th Int. Conf. on Computers and Their Applications (CATA 2005), pp. 326–332 (2005)
[SNLJ06] Schiøler, H., Nielsen, J.D., Larsen, K.G., Jessen, J.: CyNC: A method for real time analysis of systems with cyclic data flows. Journal of Embedded Computing 2(3-4), 347–360 (2006)
[SZ06] Schmitt, J.B., Zdarsky, F.A.: The DISCO network calculator - a toolbox for worst case analysis. In: Proceedings of the First International Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS 2006), Pisa, Italy. ACM, New York (November 2006)
[TCN00] Thiele, L., Chakraborty, S., Naedele, M.: Real-time calculus for scheduling hard real-time systems. In: Proc. IEEE International Symposium on Circuits and Systems (ISCAS), pp. 101–104 (2000)
[TS09] Thiele, L., Stoimenov, N.: Modular performance analysis of cyclic dataflow graphs. In: EMSOFT 2009: Proceedings of the 9th ACM International Conference on Embedded software, Grenoble, France, pp. 127–136 (2009)
[WT06] Wandeler, E., Thiele, L.: Real-Time Calculus (RTC) Toolbox (2006), http://www.mpa.ethz.ch/Rtctoolbox

A Example of Distance Between Theory and Implementation

In section 3.7, we claim that the DISCO and RTC tools implementation is not a direct mapping of the theory.

As example of this approximation, we will show that what is called "deconvolution" in the codes is not exactly the mathematical definition of the deconvolution.

To make the difference visible (*i.e.* printable), we introduce $\dot{\gamma}_{r,b}$, the left-continuous extension of $\gamma_{r,b}$ ($\dot{\gamma}_{r,b}(x) = rx + b$, and we use three specific results (with domain $\mathbb{R}_+$): $\gamma_{r,b} * \gamma_{r,b} = \gamma_{r,b}$, $\dot{\gamma}_{r,b} * \dot{\gamma}_{r,b} = \dot{\gamma}_{r,2b}$ and $\gamma_{r,b} \oslash \beta_{R,T} = \dot{\gamma}_{r,b+rT}$.

In both RTC and DISCO, when creating a γ function (with the `rtccurve` function for RTC and `createTokenBucket` for DISCO), computing its deconvolution (with resp. commands `rtcmindeconv` and `deconvolve`) by a β function (ie $f = \gamma_{r,b} \oslash \beta_{R,T}$, which can be in both tools plotted and looks like $\gamma_{r,b+rT}$ or $\dot{\gamma}_{r,b+rT}$, the continuity being hard to see on screen) computing the auto-convolution $f * f$ (with resp. commands `rtcminconv` and `convolveAlmostConcave`), we get f, when we should get $\dot{\gamma}_{r,2b+2rT}$.

What is really computed by functions with name "deconvolution" in DISCO and RTC is not really the deconvolution, but the composition of convolution and

minimum with δ_0, ie, if `deconv`(f,g) denotes the implementation in these tools, we have $\texttt{deconv}(f, g) = (f \oslash g) \wedge \delta_0$, instead of $f \oslash g$.

It is not a real bug: the computed bounds are still true, since the deconvolution in network calculus is always used for computing arrival curves, and it is known that, if f is an arrival curve for a flow, than also is $f \wedge \delta_0$. But this distance between mathematical domain and implementation does not help beginners to discover the network calculus.

DEBORAH: A Tool for Worst-Case Analysis of FIFO Tandems

Luca Bisti, Luciano Lenzini, Enzo Mingozzi, and Giovanni Stea

Dipartimento di Ingegneria dell'Informazione, University of Pisa
Via Diotisalvi 2, I-56122 Pisa, Italy
{luca.bisti,l.lenzini,e.mingozzi,g.stea}@iet.unipi.it

Abstract. Recent results on Network Calculus applied to FIFO networks show that, in the general case, end-to-end delay bounds cannot be computed in a closed form, whereas solving non-linear programming problems is instead required. Furthermore, it has been shown that these bounds may be larger than the actual worst-case delay, which also calls for computing *lower* bounds on the latter. This paper presents DEBORAH (Delay Bound Rating AlgoritHm), a tool for computing upper and lower bounds to the worst-case delay in FIFO tandem networks. DEBORAH can analyze tandems of up to several tens of nodes in reasonable time on off-the-shelf hardware. We overview the various algorithms used by DEBORAH to perform the various steps of the computations, and describe its usage.

1 Introduction

Computing *good* end-to-end delay bounds in networks employing per-aggregate resource management (such as DiffServ [2] or MPLS [4]) is a challenging task. A delay bound is as good as it is *tight*, i.e. close to the actual *worst-case delay* (WCD) for a bit of the flow being analyzed. While it is fairly simple to compute the WCD under per-flow resource management [3], this task appears to be considerably more complex in networks employing per-aggregate resource management. A recent work shows that WCD computation is actually NP-hard under *blind* multiplexing [15], i.e. where no assumption is made regarding the flow multiplexing criterion (e.g. both FIFO and strict-priority fit this definition). Tools for computing delay bounds in the above settings have also been released [12-13,15]. As for FIFO-multiplexing networks (where the FIFO hypothesis generally allows for smaller delays), our previous works [1,9-11] describe a methodology for computing per-flow delay bounds in tandem networks of rate-latency nodes traversed by leaky-bucket shaped flows. The method, called *Least Upper Delay Bound* (LUDB), is based on the well-known Network Calculus theorem that allows a parametric set of *per-flow* service curves to be inferred from a *per-aggregate* service curve at a single node. As shown in [11], we can derive end-to-end service curves only for *nested* tandems, where the path traversed by a flow *a* is either entirely included into the path of another flow *b* or has a null intersection with it. Non-nested tandems, instead, have to be *cut* into a number of nested sub-tandems, which have to be analyzed separately using LUDB. Then, *per sub-tandem* delay bounds are computed and summed up to obtain the end-to-end delay bound. In this

T. Margaria and B. Steffen (Eds.): ISoLA 2010, Part I, LNCS 6415, pp. 152–168, 2010.

case, there are always several ways for cutting a tandem, and there is no way to state *a priori* whether one leads to better results than the others. The algorithmic difficulties involved into computing the LUDB are non trivial: closed-form solutions are known only for specific topologies (e.g., tree networks [10] and tandems with 1-hop persistent interfering traffic [9]). In the other cases, the piecewise linear problem that comes out of (sub-)tandem analysis has not been proved to be convex. Moreover, the number of possible cuts for a non-nested tandem may grow exponentially with the tandem length. Therefore, software tools that allow tandem analysis are necessary even for small-scale problems. Moreover, recent research [1] has shown that the LUDB may actually be larger than the WCD. Therefore, it is also important to compute *lower bounds* on the WCD, which requires the ability to perform operations (e.g., multiplexing/demultiplexing and convolution) on cumulative arrival functions.

In this paper we present the algorithms used within DEBORAH (DElay BOund Rating AlgoritHm, [14]), an open-source C++ tool, for computing upper and lower bounds on the WCD. The algorithms are in general exponential, but – thanks to clever implementations – appears to be fast enough to manage tandems of up to several tens of nodes within a reasonable time.

The rest of the paper is organized as follows: we report some background on Network Calculus in Section 2. Section 3 reports the system model. We describe the LUDB methodology in Section 4, and Section 5 highlights the algorithms coded into DEBORAH and presents its interface. Section 6 concludes the paper.

2 Network Calculus Background

We report a minimal background on Network Calculus, essentially with the purpose of laying down the notation for the remainder. Interested readers are referred to the abundant literature for further information [3,5-6,8,16].

Let $A(t)$ and $D(t)$ be the (wide-sense increasing) *Cumulative Arrival* and *Cumulative Departure* functions (CAF, CDF) for a flow at a network element. That network element can be modeled by the *service curve* $\beta(t)$ if:

$$D(t) \geq \inf_{0 \leq s \leq t} \{A(t-s) + \beta(s)\} = (A \otimes \beta)(t), \tag{1}$$

for any $t \geq 0$. The flow is said to be guaranteed the (minimum) service curve β, which is the min-plus convolution ($\otimes$) of A and β. Min-plus convolution is commutative and associative, and convolution of concave curves with $f(0) = 0$ is equal to their minimum. Several network elements, such as delay elements, links, and regulators, can be modeled by service curves. For example, network elements which have a transit delay bounded by φ can be described by the following service curve:

$$\delta_\varphi(t) = \begin{cases} +\infty & t \geq \varphi \\ 0 & t < \varphi \end{cases}.$$

Many commonplace packet schedulers can be modeled by a family of simple service curves called the rate-latency service curves, defined as $\beta_{\theta,R}(t) = R \cdot [t-\theta]^+$ for some $\theta \geq 0$ (the latency) and $R \geq 0$ (the rate). Notation $[x]^+$ denotes $\max\{0, x\}$. A fundamental result of Network Calculus is that the service curve of a feed-forward sequence of network elements traversed by a data flow is obtained by convolving the service curves of each of the network elements.

A*rrival curves* constrain the CAFs. A wide-sense increasing function α is said to be an arrival curve for CAF A if $A(t) - A(\tau) \leq \alpha(t-\tau)$, for all $\tau \leq t$. A flow regulated by a *leaky-bucket* shaper, with *sustainable rate* ρ and *burst size* σ, is constrained by the *affine* arrival curve $\gamma_{\sigma,\rho}(t) = (\sigma + \rho \cdot t) \cdot 1_{\{t>0\}}$. Function $1_{\{expr\}}$ is equal to 1 if *expr* is true, and 0 otherwise.

End-to-end delay bounds can be derived by combining together arrival and service curves. Given a FIFO network element (or tandem thereof) characterized by a service curve β and a flow traversing it, constrained by arrival curve α, the delay is bounded by the horizontal deviation:

$$h(\alpha, \beta) \triangleq \sup_{t \geq 0} \left[\inf \{ d \geq 0 : \alpha(t-d) \leq \beta(t) \} \right]. \tag{2}$$

From (2) it follows that $\beta_1 \leq \beta_2 \Rightarrow h(\alpha, \beta_1) \geq h(\alpha, \beta_2)$. Notation $\beta_1 \leq \beta_2$ means that $\forall t \ \ \beta_1(t) \leq \beta_2(t)$. Regarding FIFO multiplexing of flows into the same service curve element, a fundamental result ([7], [3], Chapter 6) allows one to derive *per-flow* service curves from *per-aggregate* ones:

Theorem 1 (FIFO Minimum Service Curves, [3])
Consider a lossless node serving two flows, 1 and 2, in FIFO order. Assume that packet arrivals are instantaneous. Assume that the node guarantees a minimum service curve β to the aggregate of the two flows. Assume that flow 2 has α_2 as an arrival curve. Define the family of functions:

$$E(\beta, \alpha, \tau)(t) = \left[\beta(t) - \alpha_2(t-\tau) \right]^+ \cdot 1_{\{t>\tau\}} \tag{3}$$

For any $\tau \geq 0$ such that $E(\beta, \alpha, \tau)(t)$ is wide-sense increasing, then flow 1 is guaranteed the (equivalent) service curve $E(\beta, \alpha, \tau)(t)$.

Theorem 1 describes an *infinity* of equivalent service curves, each instance of which (obtained by selecting a specific value for the τ parameter), is a service curve for flow 1, provided it is wide-sense increasing. Hereafter, we omit repeating that curves are functions of time (and, possibly, of other parameters such as τ) whenever doing so does not generate ambiguity.

It has been proved in [10-11] (to which the interested reader is referred for more details and proofs) that *pseudoaffine* curves effectively describe the service received by single flows in FIFO multiplexing rate-latency nodes. We call a pseudoaffine curve one which can be described as:

$$\pi = \delta_D \otimes \left[\bigotimes_{1 \le x \le n} \gamma_{\sigma_x, \rho_x} \right] = \delta_D \otimes \left[\bigwedge_{1 \le x \le n} \gamma_{\sigma_x, \rho_x} \right], \tag{4}$$

i.e., as a multiple affine curve shifted to the right, the latter inequality stemming from the fact that affine curves are concave. We denote as *offset* the non-negative term D, and as *leaky-bucket stages* the affine curves between square brackets. We denote with ρ_π^* (*long-term rate*) the smallest sustainable rate among the leaky-bucket stages belonging to the pseudoaffine curve π, i.e. $\rho_\pi^* = \min_{x=1,\ldots,n} (\rho_x)$. A rate-latency service curve is in fact pseudoaffine, since it can be expressed as $\beta_{\theta,R} = \delta_\theta \otimes \gamma_{0,R}$. Although more general than rate-latency curves, pseudoaffine curves are still fairly easy to manage from a computational standpoint: the convolution of two pseudoaffine curves is in fact still a pseudoaffine curve. Furthermore, Theorem 1 can be specialized for the case of pseudoaffine service curves and leaky-bucket arrival curves as follows:

Corollary 2 [10]
Let π be a pseudoaffine service curve, with offset D and n leaky-bucket stages $\gamma_{\sigma_x, \rho_x}$, $1 \le x \le n$, and let $\alpha = \gamma_{\sigma,\rho}$, with $\rho_\pi^* \ge \rho$. If a node guarantees a minimum service curve π to the aggregate of the two flows, and flow 2 has α as an arrival curve, then the family of functions $\left\{ \overline{E}(\pi, \alpha, s), s \ge 0 \right\}$, with:

$$\overline{E}(\pi, \alpha, s) = \delta_{h(\alpha,\pi)+s} \otimes \left[\bigotimes_{1 \le x \le n} \gamma_{\rho_x \{s + h(\alpha,\pi) - D\} - (\sigma - \sigma_x), \rho_x - \rho} \right], \tag{5}$$

are pseudoaffine equivalent service curves for flow 1. Furthermore, set $S \triangleq \left\{ \overline{E}(\pi, \alpha, s), s \ge 0 \right\}$ includes all the equivalent service curves which are relevant for computing delay bounds.

Therefore, all the "good" performance bounds that can be found by applying Theorem 1 can also be found by applying Corollary 2. As (5) is much more compact than (3), and perfectly equivalent from the point of view of computing performance bounds, we will use the former henceforth.

3 System Model

We analyze a tandem of N *nodes*, connected by links and traversed by flows, i.e. distinguishable streams of traffic. We are interested in computing a tight end-to-end delay bound for a *tagged flow*, which traverses the whole tandem from node 1 to N. At each node, *FIFO multiplexing* is in place, meaning that all flows traversing the node are buffered in a single queue First-Come-First-Served. Furthermore, the aggregate of the flows traversing a node is guaranteed a *rate-latency* service curve, with rate R^k and latency θ^k, $1 \le k \le N$. A flow can be identified by the couple (i, j), $1 \le i \le j \le N$, where i and j are the first and last node of the tandem at which the flow is multiplexed with the aggregate. Flows are *fluid*, i.e. we assume that it is

feasible to inject and service an arbitrarily small amount of traffic at a node, and are constrained by a σ,ρ *leaky-bucket* arrival curve at their ingress node. Leaky-bucket curves are additive, hence we can safely assume that at most *one* flow exists along a path and identify it using the path as a subscript. Based on how the paths of their flows are interleaved, tandems can be either *nested* or *non-nested*. In a *nested* tandem, flows are either *nested* into one another, or they have null intersection, i.e. no two flows (i,j), (h,k) exist with $i<h\le j<k$. Fig. 1a represents a nested tandem of three nodes, where flow $(3,3)$ is nested within flow $(2,3)$. Furthermore, flows $(1,1)$, $(3,3)$ and $(2,3)$ are nested within the tagged flow $(1,3)$. The *level of nesting* $l(i,j)$ is the number of flows (h,k) into which (i,j) is nested. With reference to Fig. 1a, it is $l(1,1)=l(2,3)=2$, $l(3,3)=3$, and $l(1,3)=1$. The *level of nesting of the tandem* is the maximum level of nesting of one of its flows, i.e. the maximum number of flows crossing a single node. A tandem of N nodes has a level of nesting no greater than N, and that the maximum number of distinguishable flows in an N-node nested tandem is $2N-1$.

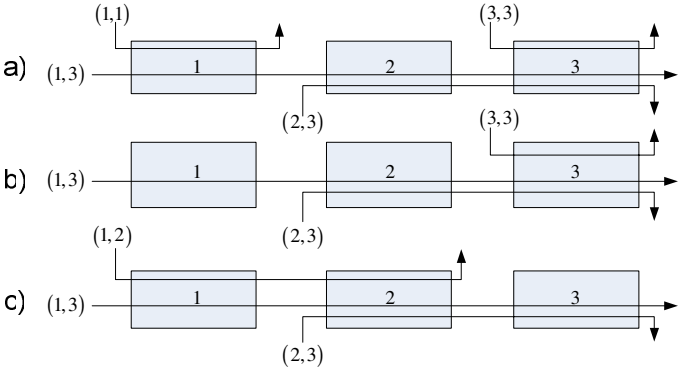


Fig. 1. a) A nested tandem; b) a fully nested tandem; c) a non-nested tandem

A particular case of n-level nested tandem is the one where there is only *one* flow at each level of nesting, i.e. $\forall x, 1\le x\le n, \exists!(i,j): l(i,j)=x$. We call such a tandem a *fully nested tandem*. A sink-tree tandem, where there are exactly N flows (i,N), $1\le i\le N$ (see Fig. 1b, above), is a fully nested tandem (whose level of nesting is N). On the other hand, a tandem is non-nested if it does not verify the above definition, as the one shown in Fig. 1c. In that case, we say that flow $(1,2)$ *intersects* flow $(2,3)$.

Finally, as far as rate provisioning is concerned, we assume that a node's rate is no less than the sum of the sustainable rates of the flows traversing it, i.e. for every node $1\le h\le N$, $\sum_{(i,j):\, i\le h\le j}\rho_{(i,j)}\le R^h$. Note that this allows a node's rate to be utilized up to 100%, and it is therefore a necessary condition for stability. Moreover, we assume that the buffer of a node is large enough as to guarantee that traffic is never dropped.

4 The LUDB Methodology

In this paragraph, we briefly describe the *Least Upper Delay Bound* (LUDB) methodology [1,11]. We first explain it on nested tandems, and extend it to non-nested tandems later on. At a first level of approximation, LUDB consists in computing *all* the service curves for the tagged flow: we start from the aggregate service curves at each node, we apply Corollary 2 iteratively in order to remove one flow $(i,j) \neq (1,N)$ from the tandem, and we convolve the service curves of nodes traversed by the same set of flows. Every time Corollary 2 is used, a new free parameter $s_{(i,j)}$ is introduced. Therefore, we compute in fact a *multi-dimensional infinity* of service curves. From each of these we can compute a delay bound for the tagged flow, hence the minimum among all the delay bounds is the *least upper delay bound*.

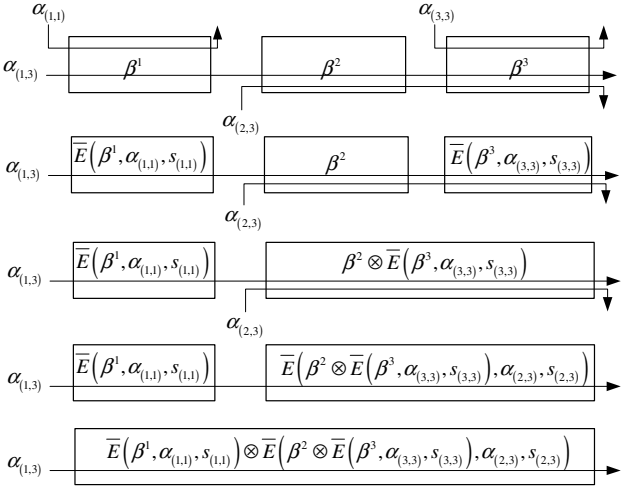


Fig. 2. An example of application of the LUDB methodology

For instance, Fig. 2 shows how to compute the set of end-to-end service curves for the tagged flow (1,3) in the nested tandem shown in Fig. 1a. The set of resulting (pseudoaffine) service curves $\pi^{\{1,3\}}$ depend on three parameters, $s_{(1,1)}$, $s_{(2,3)}$, $s_{(3,3)}$. More generally, let us consider a nested tandem of N nodes, whose level of nesting is $n \geq 2$ (otherwise the problem is trivial). The algorithm for computing the delay bound for the tagged flow is the following. As a first step, we build the *nesting tree* of the tandem, which is in fact a simplified representation of the tandem. Let us define two sets:

$$S_{(h,k)} = \{(i,j) : h \leq i \leq j \leq k \text{ and } l(i,j) = l(h,k) + 1\},$$

i.e. the set of flows which are nested right into (h,k), and:

$$C_{(h,k)} = \{l : h \leq l \leq k \text{ and } \forall (i,j) \in S_{(h,k)}, l < i \text{ or } l > j\},$$

i.e. the set of nodes in path (h,k) that are not in the path of any flow in $S_{(h,k)}$. Note that, if $S_{(h,k)} = \varnothing$, then $C_{(h,k)} = \{h, h+1, ..., k\}$.

For the sake of clarity, hereafter the nodes in the nesting tree are called *t-nodes*, in order to distinguish them from the nodes in the path of the tagged flow. In the nesting tree, there are two kind of t-nodes: non-leaf t-nodes represent *flows*, and leaf t-nodes represent *sets of nodes* in the path. More specifically, each non-leaf t-node contains a flow (h,k). The root t-node contains $(1,N)$. Furthermore, each t-node whose content is (h,k) has all flows $(i,j) \in S_{(h,k)}$ as direct descendants. Furthermore, if $C_{(h,k)} \neq \varnothing$, (h,k) has *one* more direct descendant representing $C_{(h,k)}$ (which is a leaf t-node). The level of nesting of a flow is the depth of the corresponding t-node in the nesting tree. Accordingly, we henceforth write that $(i,j) \rightarrow (h,k)$ iff $(i,j) \in S_{(h,k)}$, $S_{(h,k)}$ being the set of non-leaf *direct descendants* of (h,k), and that $(i,j) \rightarrow^{*} (h,k)$ to denote that (i,j) is a (possibly non-direct) descendant of (h,k). Fig. 3 shows the nesting tree of the tandem of Fig. 1a. Leaf t-nodes are shown as circles, while non-leaf nodes are ellipses. For instance, it is $(2,3) \rightarrow (1,3)$ and $(3,3) \rightarrow^{*} (1,3)$, whereas $(1,1) \not\rightarrow^{*} (2,3)$.

Once the nesting tree has been constructed, the set of end-to-end service curves for $(1,N)$ is computed by visiting the nesting tree from the leaves to the root as follows:

For each *leaf* t-node representing $C_{(h,k)}$ for some parent t-node (h,k), compute $\pi^{C_{(h,k)}} = \otimes_{j \in C_{(h,k)}} \beta^j$. Then, at a non-leaf t-node (h,k), compute a service curve as:

$$\pi^{\{h,k\}} = \pi^{C_{(h,k)}} \otimes \left[\bigotimes_{(i,j) \in S_{(h,k)}} \overline{E}\left(\pi^{\{i,j\}}, \alpha_{(i,j)}, s_{(i,j)}\right) \right], \tag{6}$$

i.e. as the convolution of i) the service curves obtained by applying Corollary 2 to the service curve computed at all child t-nodes, and ii) service curve $\pi^{C_{(h,k)}}$, if $C_{(h,k)} \neq \varnothing$ (otherwise assume for completeness that $\pi^{C_{(h,k)}} = \delta_0 = \beta_{0,+\infty}$).

The set of end-to-end service curves for $(1,N)$, call it $\pi^{\{1,N\}}$, is obtained by computing the service curve at the root t-node. The LUDB for the tagged flow is the following:

$$V = \min_{\substack{s_{(i,j)} \geq 0, \\ (i,j) \rightarrow^{*} (1,N)}} \left\{ h\left(\alpha_{(1,N)}, \pi^{\{1,N\}}\left(s_{(i,j)} : (i,j) \rightarrow^{*} (1,N)\right)\right) \right\}. \tag{7}$$

Since $\pi^{\{1,N\}}$ is pseudoaffine and $\alpha_{(1,N)}$ is an affine curve, problem (7) is an optimization problem with a *piecewise linear* objective function. Computing the LUDB means solving a *piecewise-linear programming* (P-LP) problem.

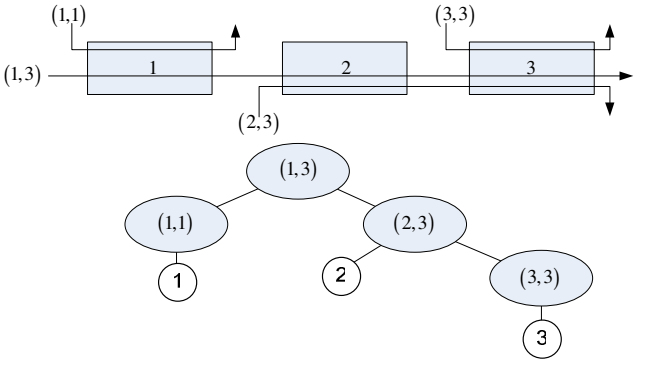


Fig. 3. A nested tandem and the related nesting tree

The LUDB methodology cannot be applied *directly* to non-nested tandems, such as the one shown in Fig. 1c. In fact, in that case, two flows intersect each other. In [11], it was observed that a non-nested tandem can always be *cut* into at most $\lceil N/2 \rceil$ *disjoint nested sub-tandems*. Therefore, one can use LUDB to compute partial, per sub-tandem delay bounds, and an end-to-end delay bound can be then computed by summing up the partial delay bounds. For instance, the tandem of Fig. 1c can be cut in two different ways, i.e. placing the cut *before* or *after* node 2. This way, two different end-to-end delay bounds can be computed, call them V^a and V^b, both using LUDB. Now, $V = V^a \wedge V^b$ is an end-to-end delay bound for the tagged flow, and both V^a and V^b can actually be the minimum, depending on the actual values of the nodes and flows parameters. As shown in [11], for each sub-tandem other than the first one, computing the *output arrival curve* for all the flows that traverse the cut is also required. For instance, if the cut is placed before node 2, the output arrival curve of flow (1,3) and (1,2) at the exit of node 1 are required in order to be able to analyze sub-tandem {2,3}. We proved in [11] that computing these output arrival curves is in fact identical to solving a simplified LUDB problem, and proposed algorithms to keep flows bundled as much as possible, for the sake of tightness. Therefore, the problem of computing bounds in non-nested tandem can be exploded into a number of LUDB computations, once an algorithm for computing *sets of cuts* is given.

5 DEBORAH

Hereafter, we overview the algorithms coded into DEBORAH [14] for computing upper bounds - in both nested and non-nested tandems - and those for computing lower bounds, which are common to both. We conclude the section by describing how to use the tool.

5.1 Nested Tandems

As shown in [11], P-LP problems such as (7) can be decomposed into a number of simplexes. Assume for ease of notation that:

$$\pi^{\{1,N\}} = \delta_D \otimes \left[\bigwedge_{1 \le x \le n} \gamma_{\sigma_x, \rho_x} \right],$$

where D and σ_x, $1 \le x \le n$, are linear functions of $s_{(i,j)}$, $(i,j) \to^* (1,N)$. Then, problem (7) can be formulated as follows:

$$V = \min \left\{ D + \bigvee_{1 \le x \le n} \left[\frac{\sigma_{(1,N)} - \sigma_x}{\rho_x} \right]^+ \right\}$$
$$s.t. \quad .$$
$$s_{(i,j)} \ge 0, \qquad \forall (i,j) \to^* (1,N)$$

The above problem has a piecewise linear objective function, due to the maximum operator. It can however be decomposed into $n+1$ problems, as many as the terms in the max operator between square brackets (i.e., all the n leaky bucket stages of $\pi^{\{1,N\}}$, plus the null term given by $[\ ]^+$). In each sub-problem, the max is assumed to be achieved either for generic term x, $1 \le x \le n$, or for the null term, and the inequalities which are required for these assumptions to hold are added accordingly. We henceforth call each of those instances a *decomposition* of the original (P-LP) LUDB problem. The generic decomposition x, $1 \le x \le n$, and the $n+1^{\text{th}}$ one are shown below:

$$V_x = \min \left\{ D + \frac{\sigma_{(1,N)} - \sigma_x}{\rho_x} \right\} \qquad \text{and} \qquad V_{n+1} = \min \{D\}$$
$$s.t. \qquad\qquad\qquad\qquad\qquad\qquad s.t. \quad .$$
$$s_{(i,j)} \ge 0, \qquad \forall (i,j) \to^* (1,N) \qquad\qquad s_{(i,j)} \ge 0, \qquad \forall (i,j) \to^* (1,N)$$
$$\frac{\sigma_{(1,N)} - \sigma_x}{\rho_x} \ge \frac{\sigma_{(1,N)} - \sigma_y}{\rho_y} \qquad \forall y, 1 \le y \le n \qquad\qquad \frac{\sigma_{(1,N)} - \sigma_y}{\rho_y} \le 0 \qquad \forall y, 1 \le y \le n$$
$$\frac{\sigma_{(1,N)} - \sigma_x}{\rho_x} \ge 0$$

Then, the LUDB is computed as $V = \min_{1 \le x \le n+1} \{V_x\}$. The offset D and the bursts σ_x of the n stages of $\pi^{\{1,N\}}$ can either be *affine* functions of $s_{(i,j)} \ge 0$, $(i,j) \to^* (1,N)$, in which case all the decompositions are simplexes, or can themselves be piecewise linear functions of $s_{(i,j)} \ge 0$, $(i,j) \to^* (1,N)$. In this last case, however, they are obtained by composing sum and maximum operations recursively according to the nesting tree structure. Therefore, each problem can be recursively decomposed into a number of other problems, working out maxima and adding constraints at each recursive step, until the resulting problems turn out to be simplexes themselves. We call each such simplex a *recursive simplex decomposition* (RSD) of that LUDB problem. The problem with this approach is that the number of RSDs grows exponentially with

the number of flows and nodes. As an example, for a case-study tandem with 30 nodes and 31 flows, nested up to level five, we obtain $\Omega = 1.5 \cdot 10^9$ RSDs.

A much more efficient solution algorithm can be obtained by observing that most RSDs are *infeasible* and can be identified as such at a small cost. In fact, thanks to the recursive structure of the LUDB problem, it is fairly easy to identify small sets of infeasible constraints, each one of which may appear in possibly many RSDs. Once a set of constraints is identified as infeasible, all the RSDs which include that set can be safely skipped, reducing the overall number of simplexes to be solved to a much smaller figure. Thus, the key to a faster solution algorithm is to work *bottom-up* in the nesting tree: starting from the t-nodes immediately above the leaves, we compute the LUDB for the *sub-tree* rooted at each of them. In doing so, we check the feasibility of each resulting RSD, and we mark infeasible constraints or sets thereof. Moving upwards towards the root, at each father t-node we solve the LUDB problem, this time considering *only* those RSDs which do not include infeasible (sets of) constraints of child t-nodes. As soon as new infeasible RSDs are identified, they are marked and ruled out from then on. The bottom-up approach is considerably faster than a brute-force recursive simplex decomposition. For instance, in sink-tree tandems it reduces the number of simplexes from $M!$ to $O\left(M^2\right)$. In the previously mentioned case study, the overall number of RSDs to be solved or proved infeasible is reduced from $1.5 \cdot 10^9$ to $1.67 \cdot 10^6$. In this last case, DEBORAH finds the solution in less than 20 minutes on a 2.4GHz Intel Core2 E6600 processor.

5.2 Non-nested Tandems

With non-nested tandems, before computing the LUDBs for each sub-tandem, we have to *cut* it into a number of *nested* sub-tandems. Flows (i, j) can be stored in a *flow matrix* F, such that $F_{i,j} = 1$ is flow (i, j) exists. Interdependencies can be efficiently located by exploring F using simple bitwise operations. For instance, for flow (i, j) the interdependent flows are the 1s in $[1, i-1] \times [i, j-1] \cup [i+1, j] \times [j+1, N]$.

Let d be the number of such dependencies. As a first step, an $N \times d$ binary *Dependency Matrix* (*DM*) is computed. Fig. 4 shows a non nested tandem and its *F* and *DM*. For each couple of interdependent flows (i, j) and (h, k), the dependency is severed if we cut the tandem at any node in $[h, j+1]$. Accordingly, the DM has a 1 at all the rows $[h, j+1]$ for that dependency. Row n of the DM is thus the set of dependencies D_n which would be severed by cutting the tandem at node n. We are interested in finding sets of cuts such that: i) the resulting sub-tandems are all nested, ii) all the dependencies are satisfied, and iii) no cut can be eliminated without violating i). We call the above *Primary Sets of Cuts* (PSCs).

PSCs are those subsets of rows (i.e., nodes) such that at least a 1 can be found for every column (i.e. dependency), and no row dominates the other, i.e. {3} and {2,4} in the example of Fig. 4. The number of PSCs grows exponentially with the number of nodes, however with a small exponent: for a *dense* tandem, i.e. one including *all* possible flows, our experiments show that this number grows as $0.73 \cdot 2^{0.4N}$, i.e. about $3 \cdot 10^3$

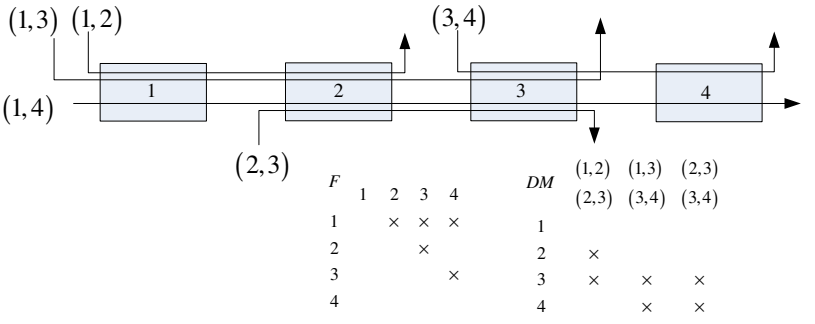


Fig. 4. A non-nested tandem, its flow matrix (left) and dependency matrix (right)

for $N = 30$. There is unfortunately no way to tell which PSC will yield the smallest bound beforehand: intuitive principles like "minimum number of cuts/flows/overall sum of ρ s" may not yield the optimal result. Hence, all PSCs should be tried in principle. While this requires to compute exponentially many LUDBs, each LUDB is also exponentially simpler because sub-tandems are generally shorter due to the cutting, hence the computation time does not differ too much with respect to the nested tandem case. Furthermore, a considerable speedup (e.g., up to 80% for dense tandems of 30 nodes) can be harvested by *caching* partial per sub-tandem computations. It turns out, in fact, that the same sub-tandem can be common to many PSCs, hence computing its partial delay bound *once* (instead of once per PSC it belongs to) improves the efficiency. The exact procedure for computing LUDBs once PSCs are known is described in [11], to which the interested reader is referred to for the details.

5.3 Lower Bounds

As proved in [1], the LUDB is not always equal to the WCD. While the two are always equal in sink-tree tandems [10], this cannot be generalized to nested tandem, nor to fully nested tandems either. Therefore, it becomes important to compute also *lower bounds* on the WCD, so as to bound the region where the WCD is included, thus implicitly assessing how overrated the LUDB can be with respect to the WCD itself. As every *achievable* delay is itself a lower bound on the WCD, we heuristically setup a scenario where delays are large, so as to create a reasonably pejorative case. The algorithm for computing the lower bound is the following:

a) assume that all nodes are lazy, i.e. that $D^{(i)} = A^{(i)} \otimes \beta^{(i)}$ for each node *i*.

b) The tagged flow $(1, N)$ sends its whole burst $\sigma_{(1,N)}$ at time $t = 0$ and then stops. Therefore, the $\sigma_{(1,N)}$ th bit of the tagged flow experiences a larger delay than the other $\sigma_{(1,N)} - 1$, and that delay is our lower bound on the WCD.

c) Every cross flow (i, j) sends "as much traffic as possible" while the bits of the tagged flow are traversing the node, so as to delay the $\sigma_{(1,N)}$ th bit of the tagged flow. Among the infinite Cumulative Arrival Functions (CAFs) that follow this rule, we

pick the *greedy* one, i.e. one such $A_{(i,j)}(t-t_0)=\alpha_{(i,j)}(t-t_0)$, t_0 being the time instant where the *first* bit of the tagged flow arrives at node *i*, and the *delayed-greedy* one, where cross flow (i,j) sends the whole burst *just before* the last bit of the tagged flow arrives. It turns out that, depending on the values associated to the nodes and flows parameters, using either greedy or delayed greedy CAFs for the cross flows actually leads to different delays, and it is not always possible to establish which is the largest beforehand.

Within DEBORAH, each CAF is represented as a piecewise linear function, not necessarily convex. This allows us to *approximate* any kind of curve. Furthermore, the curves that we use are themselves piecewise linear, so their representation is exact. Each CAF $A^k_{(i,j)}$ is a list of $Q^k_{(i,j)}$ breakpoints; each breakpoint B_i is represented through its Cartesian coordinates t_i, b *and* a *gap* g_i, i.e. a vertical discontinuity which allows for instantaneous bursts: $B_i=(t_i,b_i,g_i)$, $A^k_{(i,j)}\equiv\left\{B_x,\ 1\le x\le Q^k_{(i,j)}\right\}$. As the CAFs are wide-sense increasing, the abscissas of the breakpoints are strictly increasing, and $b_{i+1}\ge b_i+g_i$. The number of breakpoints is finite. This is because, since the worst-case delay stays finite, then it is achieved for sure in finite time, and therefore we can safely assume that the CAF remains constant after the last breakpoint. For instance, an affine CAF with initial burst σ and a constant slope ρ up to time τ is represented as $\{(0,0,\sigma),(\tau,\rho\cdot\tau+\sigma,0)\}$.

The operations required for computing the CDF of the tagged flow at node N are:

1. *FIFO multiplexing* of several CAFs at the entrance of a node, so as to compute the aggregate CAF.
2. *Convolution* between the aggregate CAF and a node's rate-latency service curve, i.e. computation of a lower bound for the aggregate CDF.
3. *FIFO de-multiplexing* of flows at the exit of a node, i.e. computation of *per-flow* CDFs from the aggregate CDF. This is required to take into account flows leaving the tandem.

The multiplexing is a summation of CAFs, which boils down to computing the union of the respective breakpoints and summing their ordinates. The convolution algorithm is explained in [3], Chapter 1.3, in its most general form. Our implementation capitalizes on the service curve being latency-rate and on the CAF being piecewise linear. In this case, all it takes is comparing the slope of the linear pieces in the CAF against the rate of the service curves, and computing intersections. The resulting CDF has a different set of breakpoints with respect to the CAF, and it is *continuous*, even if the CAF is not, since the service curve is itself continuous. The third operation, i.e. FIFO de-multiplexing, exploits the FIFO hypothesis: more specifically, for all flows (i,j) traversing node k as part of a FIFO aggregate, $\forall t\ge 0\quad D^k_{(i,j)}(t)=A^k_{(i,j)}\left(x(t)\right)$, where $x(t)=\sup\left\{\tau\le t: A^k(\tau)=D^k(t)\right\}$. Let R^{out} be the rate of $D^k(t)$ in $[t_1,t_2)$. If $A^k(t)$ is continuous in $\left[x(t_1),x(t_2)\right)$, call R^{in} and $R^{in}_{(i,j)}$ its rate and that of $A^k_{(i,j)}(t)$ in that

interval. Then, the rate of $D^k_{(i,j)}(t)$ in $[t_1, t_2)$ is equal to $R^{out}_{(i,j)} = R^{in}_{(i,j)} \cdot R^{out} / R^{in}$. If instead $A^k(t)$ has a discontinuity in t due to flow f's burst, so that $A^k(t^-) = b_0$ and $A^k(t^+) = b_1 > b_0$, then all the traffic in the aggregate CDF in $\left[\left(D^k\right)^{-1}(b_0), \left(D^k\right)^{-1}(b_1)\right)$ belongs to flow f. Thus, if R^{out} is the rate of $D^k(t)$ in that interval, the rate of $D^k_{(i,j)}(t)$ in the same interval is equal to $R^{out} \cdot 1_{\{(i,j)\equiv f\}}$. Fig. 5 reports a graphic representation of the FIFO multiplexing and de-multiplexing of two CAFs.

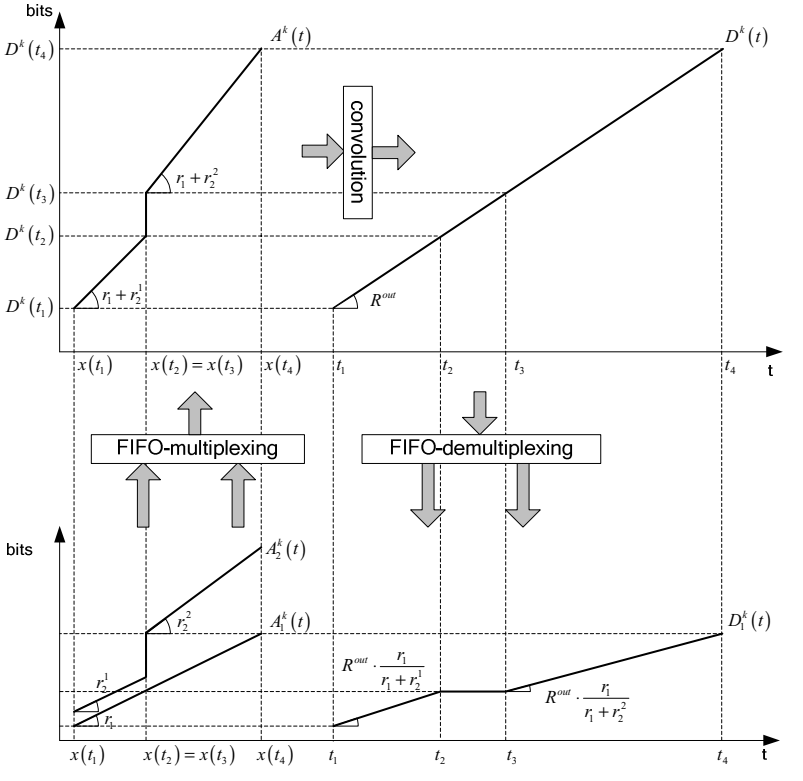


Fig. 5. Basic operations at a FIFO node

Other tools that have been developed for Network Calculus. The DISCO calculator [12] computes *output arrival curves* from (concave) input arrival curves, assuming *blind* multiplexing (instead of FIFO). The COINC library [13] implements basic (min, +) algebra operations, hence – although not a tool itself, lacking network representation, it can be used to build a tool. The RTC [18] and CyNC [19] toolboxes allow one to compute CDFs from CAFs through (min,+) convolution. However, as far as we know, they cannot compute LUDBs in FIFO systems, and we are not aware that they implement demultiplexing of flows at the output of a FIFO server, which is necessary for our lower bound analysis.

5.4 Using DEBORAH

In this section we show how to use DEBORAH for analyzing user-defined network topologies. DEBORAH is a command-line program written in portable C++, which can be compiled for a number of architectures; so far it has been successfully run on Linux, Windows and MacOS X. Its arguments can be classified into three functional categories: a) specification of the tandem topology; b) indication of the desired computation (currently LUDB, lower bound and per-node upper bound); c) network provisioning modifiers, e.g. to scale the rates assigned to flows or nodes by a constant factor or to explicitly select the tagged flow. We illustrate the most significant features offered by the tool via a couple of practical examples. As a first case we consider a nested tandem of 10 nodes and 11 flows as shown in Fig, 6. The service curves at each node and the arrival curves of each flow are specified in Table 1 and 2.

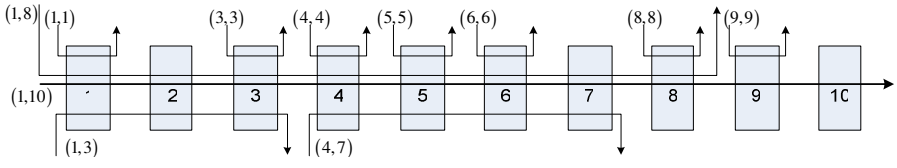


Fig. 6. Case-study nested tandem

The tandem topology is input in a text file (say "`ex1.conf`") using a straightforward syntax. The file must begin with the directive `TANDEM` N F, which denotes a tandem with N nodes and F flows. Next, the service curve of each node is configured by means of a "`NODE` n θ R" line, where n is the node ID from 1 to N, and θ, R are its latency and the rate respectively. Similarly, flows are specified using "`FLOW` i j σ ρ", where i, j are the source and sink nodes and σ, ρ are the flow's leaky bucket parameters. The tagged flow is automatically selected as the one spanning the longest segment (usually the whole tandem), or it can be manually specified by using `TFLOW` instead of `FLOW` in its declaration. Apart from the `TANDEM` directive, which is expected to come first in the file, the other lines can appear in just any order. Lines beginning with a hash (#) character are treated as comments and ignored.

A tandem configuration file is normally the first command line argument. If no other arguments are specified, DEBORAH parses the topology, performs some sanity checks (e.g. checks that the nodes' rates are sufficient) and prints a report. For nested tandems, for instance, it will print the associated nesting tree using a text notation.

The LUDB and the lower bound are computed by specifying the `-ludb` and `-lb` options after the configuration file name: `./deborah ex1.conf -ludb [-lb]`.

Regarding the LUDB, the tool reports detailed information including the numeric value of the optimal $s_{(i,j)}$ parameters and the symbolic expression of the service curve, and performance figures such as the number of simplexes evaluated and the total computation time. By default, DEBORAH runs the exact LUDB algorithm described in this paper. A heuristic approximation (not described here for reasons of space, see [17]) can be requested using the `-ludb-heuristic k` option, where *k* is the maximum number of randomly-selected RSDs used at each node in the nesting tree.

Table 1. Node provisioning

Node ID	θ, R
1	0.3, 70
2	0.2, 10
3	0.1, 40
4	0.1, 40
5	0.2, 60
6	0.1, 55
7	0.2, 7
8	0.3, 70
9	0.1, 60
10	0.2, 10

Table 2. Flow parameters

(i,j)	σ, ρ
1, 10	200, 1
1, 1	100, 60
1, 3	200, 3
3, 3	2000, 30
1, 8	100, 2
4, 4	400, 30
4, 7	300, 2
5, 5	300, 50
6, 6	200, 45
8, 8	100, 55
9, 9	300, 55

When LUDB computation is invoked, the tool detects the tandem to be non-nested and sets to cutting the tandem into multiple disjoint sub-tandems. Each computed PSC is reported in the program output along with the associated delay bound, the minimum of which is elected as the LUDB. As the critical performance factor here is represented by the possibly large number of PSCs, the latter can be controlled with the `-ludb-cuts-len L` option, which throws away PSCs exceeding the shortest one by more than *L* cuts. In fact, as *L* grows larger, a diminishing likelihood of finding good bounds can be observed. Finally, per-node bounds can be computed, using `-per-node`. The latter can be used as a baseline, as they are generally largely overrated.

For the lower bounds, DEBORAH prints the number of flow combinations analyzed versus the maximum possible, as well as the computation time. Again, the tool provides an option to deal with performance scalability issues by adding `-lb-random-combo p` to the command line, which forces the total number of combinations to be computed to stay below *p%* of the theoretical limit.

While it is easy to create and analyze custom topologies using text files, DEBORAH provides means to generate particular classes of tandems in an analytical way, which can be useful to conduct systematic studies. Specifically, it is possible to generate nested topologies whose nesting tree is a balanced trees of any order and depth, and non-nested tandems populated with an arbitrary number of flows. For the first case, the syntax is: `./deborah -gen-tree O K file.conf` where *O* is the order of the tree (the number of children for each node), *K* is the tree depth and *file.conf* is the name of the file where the configuration will be stored. Nodes and flows are provisioned according to stochastic variables which can be controlled using dedicated switches. For non-nested tandems, command `-gen-nnested N F file.conf` is used, where *N* is the number of nodes and *F* is the *percentage* of flows (randomly selected) with respect to a maximum of $N \cdot (N-1)/2$.

Finally, loaded configurations can be altered before processing takes place. For instance it is possible to override the tagged flow ID with `-tagged N`, or to scale the rates by a given factor simultaneously with `-scale-rates Rf Rn`, where the rates of flows and nodes are multiplied by *Rf* and *Rn* respectively.

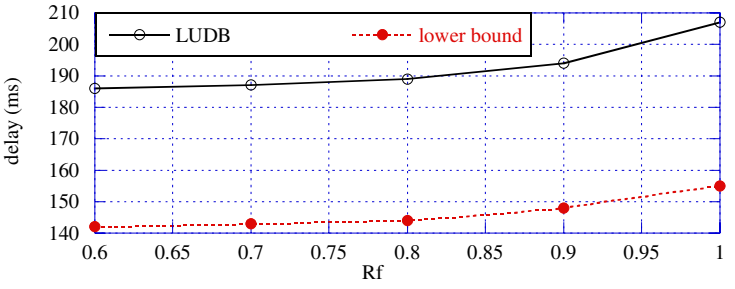


Fig. 7. LUDB / lower-bound comparison

As an example, Fig. 7 reports the LUDB and lower bound as a function of the flow rates for the nested tandem of Fig. 6. The figure was obtained using, a simple BASH script to repeatedly invoke DEBORAH with the different values for *Rf*. Computation times (subject to the coarse precision of Linux timers) are feebly, if at all, affected by the load, and they are 10ms and 40ms for the upper and lower bounds respectively.

6 Conclusions

This paper presented DEBORAH, a tool for computing upper and lower bounds on the worst-case delay in FIFO tandems using Network Calculus. We described the algorithms used for i) computing the LUDB in nested tandems, ii) cutting a non-nested tandem into nested sub-tandems, so as to enable partial LUDB computations, and iii) computing lower bounds on the worst-case delay. We have also described how to use the tool for analysis. To the best of our knowledge, DEBORAH is the first tool to compute bounds on the delay in FIFO tandem networks.
Future work on the tool will consider employing integer fractions – instead of floating points – to model slopes in the computation of the lower bounds, so as to reduce the impact of floating point arithmetic errors in the computations.

References

1. Bisti, L., Lenzini, L., Mingozzi, E., Stea, G.: Estimating the Worst-case Delay in FIFO Tandems Using Network Calculus. In: VALUETOOLS 2008, Athens, GR, October 21-23 (2008)
2. Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z., Weiss, W.: An Architecture for Differentiated Services. IETF RFC 2475 (1998)
3. Le Boudec, J.-Y., Thiran, P.: Network Calculus. LNCS, vol. 2050. Springer, Heidelberg (2001)
4. Rosen, E., Viswanathan, A., Callon, R.: Multiprotocol Label Switching Architecture. IETF RFC 3031 (January 2001)
5. Cruz, R.L.: A calculus for network delay, part i: Network elements in isolation. IEEE Transactions on Information Theory 37(1), 114–131 (1991)
6. Cruz, R.L.: A calculus for network delay, part ii: Network analysis. IEEE Transactions on Information Theory 37(1), 132–141 (1991)

7. Agrawal, R., Cruz, R.L., Okino, C., Rajan, R.: Performance Bounds for Flow Control Protocols. IEEE/ACM Trans. on Networking 7(3), 310–323 (1999)
8. Chang, C.S.: Performance Guarantees in Communication Networks. Springer, New York (2000)
9. Lenzini, L., Mingozzi, E., Stea, G.: Delay Bounds for FIFO Aggregates: a Case Study. Elsevier Computer Communications 28(3), 287–299 (2005)
10. Lenzini, L., Martorini, L., Mingozzi, E., Stea, G.: Tight End-to-end Per-flow Delay Bounds in FIFO Multiplexing Sink-tree Networks. Performance Evaluation 63, 956–987 (2006)
11. Lenzini, L., Mingozzi, E., Stea, G.: A Methodology for Computing End-to-end Delay Bounds in FIFO-multiplexing Tandems. Performance Evaluation 65, 922–943 (2008)
12. Schmitt, J.B., Zdarsky, F.A.: The DISCO Network Calculator - A Toolbox for Worst Case Analysis. In: Proc. of VALUETOOLS 2006, Pisa, Italy. ACM, New York (October 2006)
13. Bouillard, A., Thierry, É.: An Algorithmic Toolbox for Network Calculus. Journal of Discrete Event Dynamic Systems 18(1), 3/49 (2008)
14. Website of the Computer Networking Group at the University of Pisa, continuously updated, `http://cng1.iet.unipi.it/wiki/index.php/Deborah`
15. Bouillard, A., Johuet, L., Thierry, E.: Tight performance bounds in the worst-case analysis of feed-forward networks. In: Proc. INFOCOM 2010, San Diego, US, March 14-19 (2010)
16. Fidler, M.: A Survey of Deterministic and Stochastic Service Curve Models in the Network Calculus. IEEE Communications Surveys and Tutorials 12(1), 59–86 (2010)
17. Bisti, L., Lenzini, L., Mingozzi, E., Stea, G.: Computation and Tightness assessment of delay bounds in FIFO-multiplexing tandems. Technical Report, Pisa (May 2010), `http://info.iet.unipi.it/~stea/`
18. Wandeler, E., Thiele, L.: Real-Time Calculus (RTC) Toolbox,(2006), `http://www.mpa.ethz.ch/Rtctoolbox`
19. Schioler, H., Schwefel, H.P., Hansen, M.B.: CyNC – a MATLAB/Simulink Toolbox for Network Calculus. In: Proc. VALUETOOLS 2007, Nantes, FR (October 2007)

A Self-adversarial Approach to Delay Analysis under Arbitrary Scheduling

Jens B. Schmitt, Hao Wang, and Ivan Martinovic

{jschmitt,wang,martinovic}@cs.uni-kl.de

Abstract. Non-FIFO processing of flows by network nodes is a frequent phenomenon. Unfortunately, the state-of-the-art analytical tool for the computation of performance bounds in packet-switched networks, network calculus, cannot deal well with non-FIFO systems. The problem lies in its conventional service curve definitions. Either the definition is too strict to allow for a concatenation and consequent beneficial end-to-end analysis, or it is too loose and results in infinite delay bounds. Hence, in this paper, we propose a new approach to derive tight bounds in tandems of non-FIFO nodes, the so-called self-adversarial approach. The self-adversarial approach is based on a previously proposed method for calculating performance bounds in feedforward networks [30]. By numerical examples we demonstrate the superiority of the self-adversarial approach over existing methods for the analysis of non-FIFO tandems as well as that for low to medium utilizations it even stays close to corresponding FIFO performance bounds.

1 Introduction

1.1 Motivation

In the recent past, network calculus [10,25] has shown promise as an alternative methodology, besides classical queueing theory, for the performance analysis of packet-switched networks. It has found usage as a basic tool for attacking several important network engineering problems: most prominently in the Internet's Quality of Service proposals IntServ and DiffServ, but also in other environments like wireless sensor networks [22,29], switched Ethernets [31], Systems-on-Chip (SoC) [8], or even to speed-up simulations [21], to name a few. Unfortunately, it comes up short in certain fundamental aspects to really catch on as *the* system theory for the Internet as which it is sometimes advertised (see for the subtitle and the discussion in the introduction of the book by Le Boudec and Thiran [25]). Two prominent fundamental limitations that can be raised are: (1) its deterministic nature, and (2) its dependence on strict FIFO processing of flows. While still being open to some degree, the first issue has been dealt with extensively in literature, and particularly recent approaches towards a stochastic relaxation of network calculus can, for example, be found in [20,11,16]. Yet, the second issue about non-FIFO processing is still largely unexplored. In the next subsection, we provide an overview of the previous scarce work we could find on

T. Margaria and B. Steffen (Eds.): ISoLA 2010, Part I, LNCS 6415, pp. 169–183, 2010.

this topic. In contrast, non-FIFO *multiplexing* between flows was an intensive subject in previous work (see, for example, [30,7] and the references therein to recent contributions on the so-called arbitrary or general multiplexing). In this paper, we are concerned with the scheduling within a flow under analysis, so we deal with the questions *when* work units are processed by a node and *in which order*. The first question on how a node provides its capacity to a flow is flexibly answered by the network calculus concept of service curves, while the second question has so far always been answered by assuming a FIFO processing order. The goal of this paper is to provide a more flexible answer to this questions. In the following, we provide some arguments on why besides being of theoretical interest this issue should be addressed.

Assuming that the work units of a flow under analysis are processed in FIFO order constitutes a logical break for the worst-case methodology in a certain sense, as we discuss now. Assume that a flow traverses a system resulting in a certain output process. The *real delay*[1] for a work unit input at time t and output at time t' is simply defined as

$$rd(t) = t' - t.$$

Any processing order other than FIFO results in an increase of the worst-case real delay; this can be easily seen by the following argument: Assume at time t_0 a work unit which experiences the worst-case real delay is input to a FIFO system. Now assume we can change the processing order of work units. If the work unit is further delayed by scheduling work units that arrived later, then certainly the real delay of that work unit under the new processing order will be worse. Processing that work unit earlier will make its new real delay $rd'(t_0)$ smaller, yet, the work unit which was processed just ahead of the above work unit is now leaving the system when the above work unit would have left, yet that work unit has arrived at time $t_1 \leq t_0$ such that the new real delay of that work unit $rd'(t_1)$ is higher than or equal to the one from the FIFO worst-case work unit, i.e., $rd(t_0) \leq rd'(t_1)$. So, in this sense FIFO can be viewed as the best-case assumption on the processing order of the flow under analysis. Therefore, it can be seen as consistent with a worst-case methodology to release the FIFO processing assumption.

Furthermore, by providing the following real-world examples where non-FIFO behavior is exhibited, we also want to stress the practical relevance of this work:

Packet Reordering: In several studies of Internet traffic it has been shown that packet reordering is a frequent event (see, for example, [4,19]). According to these studies this is occurs because of the growing amount of parallelism on a global (use of multiple paths or parallel links) as well as on a local (device) level. In particular, for scalability reasons high-speed routers often contain a complex multi-stage switching fabric which cannot ensure to preserve the preservation of arrivals at its output. This is due to a common design trade-off where FIFO

[1] The word real is chosen for the purpose of contrasting it to the virtual delay, later on defined as delay under FIFO processing of a flow.

service at the input queues is relaxed in order to avoid head-of-line blocking by choosing from a set of packets in the input queue (often some window-based scheme is used). Furthermore, the use of link aggregation, where multiple physical lines are aggregated into a single virtual link, may often lead to a non-FIFO behavior [6].

Content-Dependent Packet Scheduling: As the last example, let us mention wireless sensor networks (WSN) where packet scheduling decisions may be based on the contents of packets following a WSN-typical data-centric paradigm. Under such circumstances hardly anything can be assumed about the scheduling order, let alone the FIFO behavior.

So, from a methodological as well as an application perspective there is a clear need for an investigation on how network calculus can be extended towards an analysis without any FIFO assumptions. Immediate questions that come up are:

- Can the existing network calculus concepts be carried over to the non-FIFO case?
- Is an efficient end-to-end analysis still possible?
- What is the cost in terms of performance bounds compared to pure FIFO systems?

1.2 Related Work

There is amazingly little existing work on the treatment of non-FIFO systems in the context of network calculus. Remarkably, in his pioneering paper [12], Cruz briefly showed how to derive a delay bound for a single work-conserving server under a general scheduling assumption (comprising any non-FIFO processing order) based on the observation that the maximum backlogged period can be bounded given that traffic is regulated. Similar results can also be found in [10]. Yet, the multiple node case as well as more general server models are not treated therein.

In [24], Le Boudec and Charny investigate a non-FIFO version of the Packet Scale Rate Guarantee ($PSRG$) node model as used in DiffServ's Expedited Forwarding definition. They show that the delay bound from the FIFO case still applies in the single node case while it does not in a specific two node case. They leave more general concatenation scenarios for further study.

In [30], we dealt with the problem of computing tight delay bounds for a network of arbitrary (non-FIFO) *aggregate multiplexers*. They show the tightness of their bounding method by sample path arguments. Yet, in contrast to the problem setting in this paper, we still make a FIFO assumption on the processing order within a flow and only allow for non-FIFO behavior between flows (see the discussion in the previous subsection). Bouillard et al. recently provided more advanced and general results for the same setting in [7], yet nevertheless, they were still based on FIFO processing per flow.

To the best of our knowledge, the only previous work that also tries to derive end-to-end delay bounds without any FIFO assumptions was done by Rizzo

and Le Boudec [27]. They investigate delay bounds for a special server model, non-FIFO guaranteed rate (GR) nodes, and show that a previously derived delay bound for GR nodes [17] is not valid for a non-FIFO case (against common belief). Furthermore, they derive a new delay bound based on the network calculus results. Their delay bound no longer exhibits the nice pay-bursts-only-once phenomenon. Based on sample path arguments, they argue that their bound is tight and thus conclude that "pay bursts only once does not hold for non-FIFO guaranteed rate nodes". In contrast, we show that non-FIFO systems may still possess a concatenation property. This seeming contradiction is discussed in more detail at the very end of this paper.

1.3 Contributions

In this work, the following contributions are made:

- We demonstrate difficulties with existing service curve definitions under non-FIFO processing.
- We introduce a new approach, called self-adversarial, that enables a true end-to-end analysis for non-FIFO systems.
- We show that, somewhat contrary to the results presented in literature, the pay-bursts-only-once phenomenon still holds for non-FIFO systems.

2 Preliminaries on Network Calculus

Network calculus is a min-plus system theory for deterministic queueing systems that builds upon the calculus for network delay in [12], [13]. The important concept of service curve was introduced in [2,9,14,23,28]. The service curve based approach facilitates the efficient analysis of tandem queues where a linear scaling of performance bounds in the number of traversed queues is achieved as elaborated in [11] and also referred to as pay-bursts-only-once phenomenon in [25]. A detailed treatment of min-plus algebra and of network calculus can be found in [3] and [10], [25], respectively.

As network calculus is built around the notion of cumulative functions for input and output flows of data, the set $\mathcal{F}$ of real-valued, non-negative, and wide-sense increasing functions passing through the origin plays a major role:

$$\mathcal{F} = \left\{ f : \mathbb{R}^+ \to \mathbb{R}^+, \forall t \geq s : f(t) \geq f(s), f(0) = 0 \right\}.$$

In particular, the input function $F(t)$ and the output function $F'(t)$, which cumulatively count for the number of work units that are input to, respectively output from, a system $\mathcal{S}$, are in $\mathcal{F}$. Throughout the paper, we assume in- and output functions to be continuous in both time and space. Note that this is not a general limitation as there exist transformations between discrete and continuous models [25].

There are two important min-plus algebraic operators:

Definition 1. *(Min-plus Convolution and Deconvolution) The min-plus convolution and deconvolution of two functions $f, g \in \mathcal{F}$ are defined to be*

$$(f \otimes g)(t) = \inf_{0 \le s \le t} \{f(t-s) + g(s)\},$$

$$(f \oslash g)(t) = \sup_{u \ge 0} \{f(t+u) - g(u)\}.$$

It can be shown that the triple $(\mathcal{F}, \wedge, \otimes)$, where $\wedge$ denotes the minimum operator (which ought to be taken pointwise for functions), constitutes a dioid [25]. Also, the min-plus convolution is a linear operator on the dioid $(\mathbb{R} \cup \{+\infty\}, \wedge, +)$, whereas the min-plus deconvolution is not. These algebraic characteristics result in a number of rules that apply to those operators, many of which can be found in [25], [10]. Let us now turn to the performance characteristics of flows that can be bounded by network calculus means:

Definition 2. *(Backlog and Virtual Delay) Assume a flow with input function F that traverses a system $\mathcal{S}$ resulting in the output function F'. The* backlog *of the flow at time t is defined as*

$$b(t) = F(t) - F'(t).$$

The virtual delay *for a work unit input at time t is defined as*

$$vd(t) = \inf \{\tau \ge 0 : F(t) \le F'(t+\tau)\}.$$

So, this is the point where the FIFO assumption sneaks in the network calculus as far as delay is concerned, because $rd(t) = vd(t)$ for all t only under FIFO processing of the flow. We use the usual network calculus terminology of the so-called *virtual* delay in contrast to the *real* delay, as defined above (see Section 1.1). Next, arrival and departure processes specified by input and output functions are bounded based on the central network calculus concepts of arrival and service curves:

Definition 3. *(Arrival Curve) Given a flow with input function F, a function $\alpha \in \mathcal{F}$ is an arrival curve for F iff*

$$\forall t, s \ge 0, s \le t : F(t) - F(t-s) \le \alpha(s) \Leftrightarrow F = F \otimes \alpha.$$

A typical example of an arrival curve is given by an affine arrival curve $\gamma_{r,b}(t) = b + rt$, $t > 0$ and $\gamma_{r,b}(t) = 0$, $t \le 0$, which corresponds to token-bucket traffic regulation.

Definition 4. *(Service Curve – SC) If the service provided by a system $\mathcal{S}$ for a given input function F results in an output function F' we say that $\mathcal{S}$ offers a service curve β iff*

$$F' \ge F \otimes \beta.$$

For continuous functions F and β this is equivalent to the following condition

$$\forall t \ge 0 : \exists s \le t : F'(t) \ge F(s) + \beta(t-s).$$

A typical example of a service curve is given by a so-called rate-latency function $\beta_{R,T}(t) = R(t-T) \cdot 1_{\{t>T\}}$, where $1_{\{cond\}}$ is 1 if the condition *cond* is satisfied and 0 otherwise. Also, nodes operating under a delay-based scheduler and guaranteeing that a work unit arriving at any time t will leave the node at time $t' \leq t + T$ for some fixed $T > 0$, i.e. $\forall t \geq 0 : rd(t) \leq T$, are known to provide a service curve $\delta_T = \infty \cdot 1_{\{t>T\}}$. We also call these *bounded latency* nodes.

Using those concepts it is possible to derive *tight* performance bounds on backlog, *virtual* delay and output:

Theorem 1. *(Performance Bounds) Consider a system $\mathcal{S}$ that offers a service curve β. Assume a flow F traversing the system has an arrival curve α. Then we obtain the following performance bounds:*

$$\textit{backlog: } \forall t : b(t) \leq (\alpha \oslash \beta)(0) =: v(\alpha, \beta),$$
$$\textit{virtual delay: } \forall t : vd(t) \leq \sup_{t \geq 0} \inf \{\tau \geq 0 : \alpha(t) \leq \beta(t+\tau)\} =: h(\alpha, \beta),$$
$$\textit{output (arrival curve } \alpha' \textit{ for } F'\textit{): } \alpha' = \alpha \oslash \beta.$$

Here, note again that the delay bound is only a *virtual* one, meaning that it is based on the FIFO assumption for the flow under analysis. One of the strongest results of the network calculus is the concatenation theorem that enables us to investigate tandems of systems as if they were single systems:

Theorem 2. *(Concatenation Theorem for Tandem Systems) Consider a flow that traverses a tandem of systems $\mathcal{S}_1$ and $\mathcal{S}_2$. Assume that $\mathcal{S}_i$ offers a service curve β_i to the flow. Then the concatenation of the two systems offers a service curve $\beta_1 \otimes \beta_2$ to the flow.*

Using the concatenation theorem, it is ensured that an end-to-end analysis of a tandem of servers achieves tight performance bounds, which in general is not the case for an iterative per-node application of Theorem 1.

3 Conventional Network Calculus and Non-FIFO Systems

In this section, we investigate how the existing network calculus can cope with non-FIFO systems. The crucial aspect is the node model. We start with the typical service curve model as defined in the previous section and then turn to strict service curves, only to find out that both of them encounter problems under non-FIFO processing.

3.1 Using Service Curves (SC) for Non-FIFO Systems

As the *SC* definition bears the advantages that many systems belong to that class and that it possesses a concatenation property, it is worthwhile an attempt to apply it also in the case of non-FIFO systems. Yet, the following example shows that it is impossible to bound the real delay in non-FIFO systems solely based on the *SC* definition:

Example 1. (*SC* Cannot Bound the Real Delay) Assume a single node system $\mathcal{S}$ which offers a rate-latency service curve $\beta = \beta_{2,1}$ to a flow F which is constrained by an affine arrival curve $\alpha = \gamma_{1,0}$. Now assume the flow to be greedy, that means $F = \alpha$ and the server to be lazy, that means $F' = F \otimes \beta$. Thus, we obtain

$$\begin{aligned} F' &= \alpha \otimes \beta = \gamma_{1,0} \otimes \beta_{2,1} = \gamma_{1,0} \otimes \gamma_{2,0} \otimes \delta_1 \\ &= (\gamma_{1,0} \wedge \gamma_{2,0}) \otimes \delta_1 \leq \gamma_{1,0} \otimes \delta_1 < \gamma_{1,0} = F. \end{aligned}$$

Hence, $\forall t \geq 0 : F'(t) < F(t)$, or equivalently, $\forall t \geq 0 : b(t) > 0$, which means that the system remains backlogged the entire time. Therefore, without any assumptions on the processing order, a certain work unit can, under these circumstances, be kept forever in the system. Thus, the real delay of that work unit is unbounded. Note that using the standard FIFO processing assumption, we can of course bound the real delay of the system by $\forall t \geq 0 : rd(t) = vd(t) \leq \frac{3}{2}$.

From this example, we see that the *SC* property is too weak as a node model for analyzing non-FIFO systems. Therefore, it is sensible to look for more stringent node models, as it is done in the following subsection.

3.2 Using Strict Service Curves (S^2C) for Non-FIFO Systems

A number of systems provides more stringent service guarantees than captured by *SC*, fulfilling the so-called strict service curve [25] (also known as strong service curve [15,2] and related to the universal service curve concept in [26])

Definition 5. *(Strict Service Curve – S^2C) Let $\beta \in \mathcal{F}$. System $\mathcal{S}$ offers a* strict *service curve β to a flow, if during any backlogged period of duration u the output of the flow is at least equal to $\beta(u)$. A backlogged period of duration u at time t is defined by the fact that $\forall s \in (t-u, t] : b(s) > 0$.*

Note that any node satisfying S^2C also satisfies *SC*, but not vice versa. For example, a bounded latency node does not provide δ_T as a *strict* service curve. In fact, it does not provide any S^2C apart from the trivial case $\beta = 0$. On the other hand, there are many schedulers that offer strict service curves; for example, most of the generalized processor sharing-emulating schedulers (e.g., PGPS [26], WF^{2}Q [5], or round robin schedulers like SRR [18], to name a few), offer a strict service curve of the rate-latency type.

Now for bounding the real delay under S^2C: In fact, as was already shown by Cruz [12] (and can also be found in [10] (Lemma 1.3.2)), the intersection point between an arrival and a *strict* service curve constitutes a bound on the length of the maximum backlogged period and thus also a bound on the real delay for such a system:

Theorem 3. *(Real Delay Bound for Single S^2C Node) Consider a system $\mathcal{S}$ that offers a strict service curve β. Assume a flow F traversing the system has an arrival curve α. Then we obtain the following bound on the real delay:*

$$rd(t) \leq \sup\{s \geq 0 : \alpha(s) \geq \beta(s)\} =: i(\alpha, \beta).$$

So, the situation has improved in comparison to the SC case: Based on the single node result one can conceive, for the multiple node case, an iterative application of Theorem 3 together with the output bound from Theorem 1. More specifically, if a tandem of n S^2C non-FIFO nodes, each providing a strict service curve $\beta_j, j = 1, \ldots, n$, is to be traversed by an α-constrained flow then a bound on the real delay can be calculated as

$$rd(t) \leq \sum_{j=1}^{n} i(\alpha \oslash \bigotimes_{k=1}^{j-1} \beta_k, \beta_j).$$

Setting for example $\beta_j = \beta_{R,T}, j = 1, \ldots, n$ and $\alpha = \gamma_{r,b}$ this results in

$$rd(t) \leq \frac{n(b + RT) + \frac{n}{2}(n-1)rT}{R - r}. \tag{1}$$

Here, we see a typical drawback of additive bounding methods, with the burst of the traffic being paid n times as well as a quadratic scaling of the bound in the number of nodes [11,25]. The key to avoid this behavior is to perform an end-to-end analysis based on a concatenation theorem. Yet, as known and demonstrated in the next example, S^2C does not possess such a concatenation property.

Example 2. (S^2C Possesses No Concatenation Property) Assume two systems $\mathcal{S}_1$ and $\mathcal{S}_2$, both providing a strict rate-latency service curve $\beta^i = \beta_{1,1}, i = 1, 2$, which are traversed in sequence by a flow F. Let F_1' and F_2' be the output functions from $\mathcal{S}_1$ and $\mathcal{S}_2$, respectively. As a candidate strict service curve for the composite system, we consider $\beta^{1,2} = \beta^1 \otimes \beta^2 = \beta_{1,2}$.

We now construct a backlogged period $[t_1, t_2]$ of the composite system such that

$$F_2'(t_2) - F_2'(t_1) < \beta^{1,2}(t_2 - t_1).$$

thereby showing that $\beta^{1,2}$ is not a strict service curve for the composite system:

Let $t_1 = 0$ and $t_2 = 3$ and assume the following behavior of the input and output function

$$F(t) = \begin{cases} \epsilon & 0 < t < 2 \\ 2\epsilon & 2 \leq t \leq 3 \end{cases}, \qquad F_1'(t) = \begin{cases} 0 & 0 \leq t \leq 1 \\ \epsilon & 1 < t \leq 3 \end{cases},$$

$$F_2'(t) = \begin{cases} 0 & 0 \leq t \leq 2 \\ \epsilon & 2 < t \leq 3 \end{cases},$$

with any $\epsilon > 0$. It is easy to check that the composite system is continuously backlogged during $[0, 3]$ as well as that each individual system is not violating its strict service curve property. Nevertheless, for any choice of $\epsilon < 1$ we obtain

$$F_2'(3) - F_2'(0) = \epsilon < \beta^{1,2}(3) = 1,$$

which shows that $\beta^{1,2}$ is not S^2C for the composite system (while, of course, being SC for it). In fact, by extending the example appropriately it can be

shown that the only strict service curve that can be guaranteed by the composite system is the trivial case $\beta = 0$. This can be seen by making ϵ arbitrarily small and alternating between backlogged and idle periods of the individual systems sufficiently often. Another way to view this, is that the backlogged period of a composite system cannot be bounded based on the individual systems providing a strict service curve.

4 The Self-adversarial Approach

In this section, we devise an approach, called the self-adversarial method, to compute a tight delay bound for non-FIFO systems based on a technique that was introduced in [30].

4.1 The Self-adversarial Method

As briefly discussed in Section 1.2, in [30], we proposed a technique for computing tight delay bounds in the network of arbitrary (non-FIFO) *aggregate multiplexers*, yet we still made a FIFO processing order assumption per flow. So, this technique is not directly applicable when releasing all FIFO assumptions and besides arbitrary multiplexing also assumes arbitrary scheduling within a flow. Nevertheless, there is a way to exploit the proposed method for the problem at hand by transforming the arbitrary scheduling problem into an arbitrary aggregate multiplexing problem. More specifically, we split the original flow, with the arrival curve α, into two sub-flows: one with the arrival curve $\alpha_1 = \gamma_{0,\epsilon}$ and the other one with the arrival curve $\alpha_2 = \alpha - \gamma_{0,\epsilon}$. Both flows traverse the same servers as the original flow. This transformation is illustrated in Figure 1.

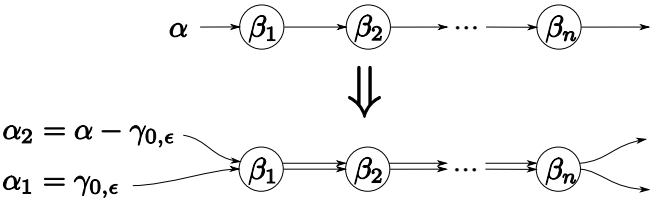


Fig. 1. Transformation of the pure non-FIFO problem into an arbitrary aggregate multiplexing problem

Now the method from [30] allows us to find the maximum left-over end-to-end service curve under arbitrary multiplexing, i.e., under any possible interleaving of the two sub-flows. To that end, the problem is reformulated as an optimization problem that can be solved by using standard methods. In [30,7], it is shown that this approach achieves tight delay bounds. So, in our case we can proceed with the following steps:

1. Computation of the left-over service curve for sub-flow 1 according to [30]: $\beta_1^{l.o.}$.
2. Computation of the delay bound for sub-flow 1: $d_1(\epsilon) = h\left(\alpha_1, \beta_1^{l.o.}\right)$.
3. Letting the delay bound for sub-flow 1 go to the limit: $d = \lim_{\epsilon \to 0} d_1(\epsilon)$.

What is effectively done here, is to assume that a part of the flow pretends to be an adversary to the other part of the flow when it comes to competition for the forwarding resources. This is why we call it the *self-adversarial* method. Taking this behavior to the limit, i.e., making the adversary part as large as possible, gives us a real delay bound as experienced by a single (infinitesimally small) work unit.

We remark that the computation of the horizontal deviation in step 2 implicitly makes a FIFO assumption for sub-flow 1. Yet, in the limit this is not a problem because a single work unit provides no degrees of freedom for the processing order any more.

Note that for the splitting of the original flow into two sub-flows we assumed that $\epsilon > 0$ is chosen such that $\alpha_2 \geq 0$. In fact, for some arrival curves this may not be possible. More precisely, if $\alpha(t)$ is continuous at $t = 0$ (e.g., a constant rate arrival curve), then the splitting described is not feasible. In such cases, the original arrival curve should be shifted to the left by some small amount Δ and set to zero for $t \leq 0$. The approach is then performed on this new (strictly larger) arrival curve. To find the delay bound under the original arrival curve, one lets $\Delta \to 0$. We decided to neglect this (rarely occurring) technicality in the above description of the self-adversarial method in order not to (further) complicate it.

4.2 Self-adversarial vs. Additive Bounding Method

Let us investigate by a simple example how the self-adversarial method works and also compare it to an additive bounding based on S^2C. Assume a token-bucket arrival curve $\gamma_{r,b}$ for the flow under investigation ($b > 0$), which traverses two servers providing *strict* rate-latency service curves $\beta_{R_i T_i}, i = 1, 2$. According to the additive bounding based on S^2C the delay bound then becomes:

$$d^{AD} = T_1 + T_2 + \frac{b + rT_1}{R_1 - r} + \frac{b + r(T_1 + T_2)}{R_2 - r}.$$

For the self-adversarial method we first split the flow into two sub-flows: subflow 1 with $\gamma_{0,\epsilon}$ and subflow 2 with $\gamma_{r,b-\epsilon}$ as arrival curves. Proceeding with the steps described in the previous section we obtain the following delay bound:

1. Computation of the left-over service for sub-flow 1 according to [30]:

$$\beta_1^{l.o.} = \beta_{\min\{R_1,R_2\}-r,\, T_1+T_2+\frac{b-\epsilon+rT_1}{\min\{R_1,R_2\}-r}+\frac{rT_2}{R_2-r}}.$$

2. Computation of the delay bound for sub-flow 1:

$$d_1(\epsilon) = \frac{\epsilon}{\min\{R_1, R_2\} - r} + T_1 + T_2 + \frac{b - \epsilon + rT_1}{\min\{R_1, R_2\} - r} + \frac{rT_2}{R_2 - r}.$$

3. Letting the delay bound for sub-flow 1 go to the limit ($\epsilon \to 0$):

$$d^{SA} = T_1 + T_2 + \frac{b + rT_1}{\min\{R_1, R_2\} - r} + \frac{rT_2}{R_2 - r}.$$

A simple inspection shows that $d^{SA} \leq d^{AD}$, where equality only holds if $b = 0 \wedge (T_1 = 0 \vee r = 0)$, which are strong restrictions. Hence, this demonstrates that the additive method is not tight under most circumstances. Similar problems with purely min-plus algebraic methods are reported and extensively discussed in [30]. These problems are inherent in using the min-plus algebraic approach. In particular, by the application of a min-plus convolution the knowledge on the *order of servers* is lost. Yet, this order is crucial to derive tight delay bounds for non-FIFO systems. The min-plus algebraic approach automatically maps a tandem of system to the worst-case order it could be in (see [30] for more discussions along this line).

So, with respect to the tightness of the computed bounds, the self-adversarial method is superior to the additive method. A potential drawback for the self-adversarial method is that the computational effort for the self-adversarial method can become very high. In particular, if arrival and service curves are piecewise-linear functions then a set of optimization problem needs to be solved first before the final left-over service curve can be constructed according to [30] (in [7] a more efficient and provably tight approach is proposed, on which the self-adversarial method could also be based). The cardinality of that set grows exponentially in the number of nodes traversed and may quickly become prohibitive. For details see [30], or even better [7], which also demonstrates the problem of computing a tight delay bound under arbitrary multiplexing in general feedforward networks to be NP-hard.

We also remark that the self-adversarial method requires S^2C servers (as in other cases like, e.g., fixed priority schedulers or arbitrary multiplexing scenarios). This requirement is crucial for setting up the optimization problem in [30] and a relaxation towards only assuming SC seems infeasible. This means, in particular, that bounded latency nodes cannot be analyzed. Similarly, according to [30], the self-adversarial method can only be applied to piecewise-linear concave arrival and convex service curves. Such a restriction does not apply, in principle, to the additive bounding method.

While the tightness of the self-adversarial method is "inherited" from [30], it can also be understood in the original system. In particular, if the processing order applied is to always choose the work unit that has entered the *network* last (assuming work units are time-stamped when they enter the network) then we conjecture that the bound can actually be achieved. This processing order has also been coined shortest-in-system (SIS) in the realm of adversarial queueing theory [1]. If only one node is traversed, then SIS becomes LIFO and clearly constitutes the worst-case processing order. In multi-node scenarios, we conjecture that SIS produces a worst-case sample path if greedy arrivals (exactly following the arrival curve) and lazy servers (exactly following the service curve) are assumed.

As the last remark, we note that if there is also cross-traffic from other flows we can first apply [30] to derive a left-over service curve for the flow of interest and then apply the self-adversarial method to arrive at tight bounds under arbitrary multiplexing *and* scheduling, i.e., a completely non-FIFO scenario.

5 Numerical Experiments

To give some feeling for the improvements achievable by using the self-adversarial approach compared to an additive bounding based on S^2C we provide some numerical experiments. In addition, we demonstrate what cost is incurred when releasing the FIFO assumption. For these numerical experiments we use simple settings: as arrival curve for the flow to be analyzed we assume a token bucket $\gamma_{r,b}$ where we set $r = 10[Mbps]$ and $b = 5[Mb]$ (unless we vary the rate r to achieve a certain utilization); for the service curves of the nodes to be traversed we use a rate-latency function $\beta_{R,T}$ with $R = 20[Mbps]$ and $T = 0.01[s]$. Unless we use the number of nodes as a primary factor in the experiments we assume $n = 10$ nodes to be traversed by the flow under investigation.

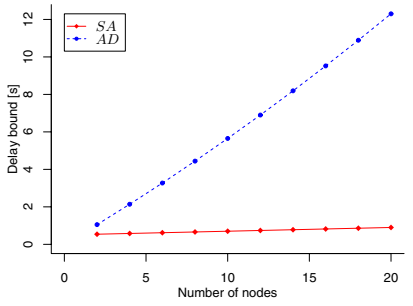
5.1 Comparison of Self-adversarial and Additive Bounding

In this first set of numerical experiments we investigate how the self-adversarial (SA) and additive (AD) bounding methods compare to each other. In Figure 2(a) the two methods are shown for a varying number of nodes (from 2 to 20). To emphasize the bad scaling of the additive method we also provide results for the same experiment with a larger number of nodes to be traversed (up to 100) in Figure 2(b). In both graphs it is obvious that the end-to-end analysis facilitated by the self-adversarial approach is highly superior and scales linearly with the number of nodes, whereas the additive bounding method scales quadratically with the number of nodes traversed and thus becomes a very conservative bound quickly.

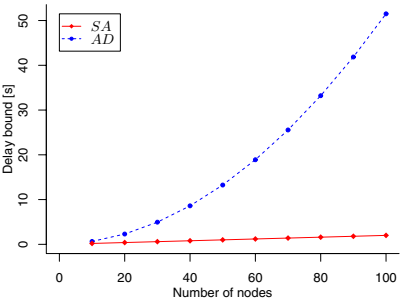
A different view on the relative performance of self-adversarial and additive methods is provided in Figure 2(c). Here, the acceptable utilizations (captured by the ratio of the the rate for the flow under investigation and the service rate of the tandem, i.e., $\frac{r}{R}$) for a given delay bound are shown for both methods. This information can be used for admission control purposes. Again, as can be clearly seen, the self-adversarial method outperforms the additive bounding by far, especially for lower delay bounds. For example, if we desire a delay bound of 2s, then an admission control using the additive bounding would return with an infeasible reply, whereas under the self-adversarial approach we could admit traffic up to $\approx 80\%$ of the service rate.

5.2 FIFO vs. Non-FIFO Delay Bounds

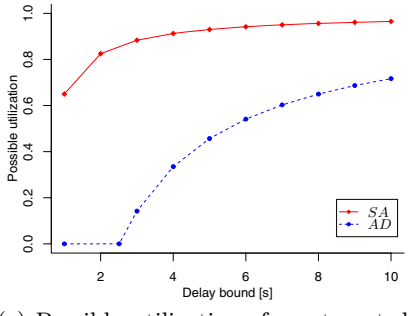
In the next set of numerical experiments, we investigate the cost of releasing the FIFO assumption in terms of delay bounds. For that purpose, we vary the utilization by increasing the sustained rate of the traffic flow under investigation (while at the same time scaling the bucket depth proportionally). As we can



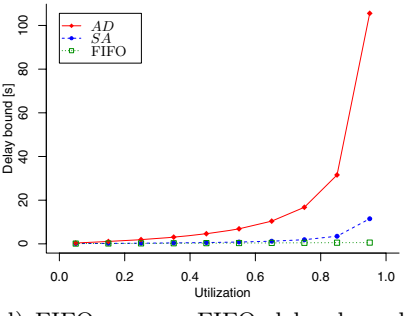
(a) Delay bounds for self-adversarial and additive bounding methods



(b) Exposing the quadratic scaling of the additive bound



(c) Possible utilizations for a target delay bound under SA and AD



(d) FIFO vs. non-FIFO delay bounds depending on the utilization

Fig. 2. Comparison of self-adversarial approach to other analysis methods under different metrics. Subfigures (a) and (b) show results for 50% utilization.

observe from Figure 2(d), only for higher utilizations there is a significant difference between the FIFO and non-FIFO delay bounds if using the self-adversarial bounding approach. On the other hand, if the additive bounding was used, the cost of releasing FIFO assumptions is high, which may be why FIFO behavior is often assumed a necessary condition to achieve good delay bounds [27]. Yet, under strict service curve assumptions and using the self-adversarial approach this assumption is not necessarily required any more.

From an application perspective, the bottom line is that only for highly utilized systems it is necessary to enforce a FIFO behavior, as far as delay bounds are concerned. For systems with lower utilizations, optimizations such as for example link aggregation or multi-stage switching fabrics do not incur a high cost in terms of worst-case delay bounds.

6 Conclusion and Discussion

In this paper, it was our goal to extend the scope of network calculus towards non-FIFO systems, as non-FIFO behavior is a reality in many networking

scenarios. It turned out that the existing service curve definitions are not satisfying under non-FIFO scheduling: they are either too loose to enable any bounding or too strict to allow for an efficient end-to-end analysis. Therefore, we devised a new approach, called the self-adversarial bounding method, which is based on previous work of ours and is provably tight. By numerical examples, we showed that the self-adversarial approach is far superior to existing methods.

The self-adversarial approach allows to recover the pay-bursts-only-once phenomenon for non-FIFO systems, which had been disputed to be valid under non-FIFO scheduling in literature [27]. This seeming contradiction is due to different assumptions on the service provided by nodes, guaranteed rate as in [27], or strict service curve, as in this paper. Since the concatenation of guaranteed rate nodes is based on their equivalence to rate-latency service curves (modulo packetization effects), a convolution of them only provides an SC guarantee and thus cannot bound the real delay, as discussed in Section 3. Hence, the only resort is an additive bounding which, however, cannot recover the pay-bursts-only-once phenomenon for the arbitrary scheduling case.

References

1. Raghavan, P., Sudan, M., Borodin, A., Kleinberg, J.M., Williamson, D.P.: Adversarial queuing theory. Journal of the ACM 48(1) (2001)
2. Agrawal, R., Cruz, R.L., Okino, C., Rajan, R.: Performance bounds for flow control protocols. IEEE/ACM Transactions on Networking 7(3), 310–323 (1999)
3. Baccelli, F., Cohen, G., Olsder, G.J., Quadrat, J.-P.: Synchronization and Linearity: An Algebra for Discrete Event Systems. Probability and Mathematical Statistics. John Wiley & Sons Ltd., Chichester (1992)
4. Bennett, J.C.R., Partridge, C., Shectman, N.: Packet reordering is not pathological network behavior. IEEE/ACM Trans. Netw. 7(6), 789–798 (1999)
5. Bennett, J.C.R., Zhang, H.: WF^2Q: Worst-case fair weighted fair queueing. In: Proc. IEEE INFOCOM, pp. 120–128 (March 1996)
6. Blanquer, J.M., Özden, B.: Fair queuing for aggregated multiple links. SIGCOMM Comput. Commun. Rev. 31(4), 189–197 (2001)
7. Bouillard, A., Jouhet, L., Thierry, E.: Tight performance bounds in the worst-case analysis of feed-forward networks. In: Proc. IEEE INFOCOM, pp. 1–9 (March 2010)
8. Chakraborty, S., Kuenzli, S., Thiele, L., Herkersdorf, A., Sagmeister, P.: Performance evaluation of network processor architectures: Combining simulation with analytical estimation. Computer Networks 42(5), 641–665 (2003)
9. Chang, C.-S.: On deterministic traffic regulation and service guarantees: A systematic approach by filtering. IEEE Transactions on Information Theory 44(3), 1097–1110 (1998)
10. Chang, C.-S.: Performance Guarantees in Communication Networks. In: Telecommunication Networks and Computer Systems. Springer, Heidelberg (2000)
11. Ciucu, F., Burchard, A., Liebeherr, J.: A network service curve approach for the stochastic analysis of networks. In: Proc. ACM SIGMETRICS, pp. 279–290 (June 2005)
12. Cruz, R.L.: A calculus for network delay, Part I: Network elements in isolation. IEEE Transactions on Information Theory 37(1), 114–131 (1991)

13. Cruz, R.L.: A calculus for network delay, Part II: Network analysis. IEEE Transactions on Information Theory 37(1), 132–141 (1991)
14. Cruz, R.L.: Quality of service guarantees in virtual circuit switched networks. IEEE Journal on Selected Areas in Communications 13(6), 1048–1056 (1995)
15. Cruz, R.L., Okino, C.M.: Service guarantees for window flow control. In: Proc. 34th Allerton Conf. Communications, Control, and Computing (October 1996)
16. Fidler, M.: An end-to-end probabilistic network calculus with moment generating functions. In: Proc. of IEEE IWQoS, pp. 261–270 (June 2006)
17. Goyal, P., Lam, S.S., Vin, H.M.: Determining end-to-end delay bounds in heterogeneous networks. Multimedia Syst. 5(3), 157–163 (1997)
18. Guo, C.: SRR: An O(1) time complexity packet scheduler for flows in multi-service packet networks. IEEE/ACM Transactions on Networking 12(6), 1144–1155 (2004)
19. Jaiswal, S., Iannaccone, G., Diot, C., Kurose, J., Towsley, D.: Measurement and classification of out-of-sequence packets in a tier-1 IP backbone. IEEE/ACM Trans. Netw. 15(1), 54–66 (2007)
20. Jiang, Y.: A basic stochastic network calculus. In: Proc. ACM SIGCOMM, pp. 123–134 (September 2006)
21. Kim, H., Hou, J.C.: Network calculus based simulation: theorems, implementation, and evaluation. In: Proc. IEEE INFOCOM (March 2004)
22. Koubaa, A., Alves, M., Tovar, E.: Modeling and worst-case dimensioning of cluster-tree wireless sensor networks. In: Proc. of RTSS 2006, Rio de Janeiro, Brazil, pp. 412–421. IEEE Computer Society, Los Alamitos (2006)
23. Le Boudec, J.-Y.: Application of network calculus to guaranteed service networks. IEEE Transactions on Information Theory 44(3), 1087–1096 (1998)
24. Le Boudec, J.-Y., Charny, A.: Packet scale rate guarantee for non-fifo nodes. In: Proc. IEEE INFOCOM, pp. 23–26 (June 2002)
25. Le Boudec, J.-Y., Thiran, P.: Network Calculus. LNCS, vol. 2050. Springer, Heidelberg (2001)
26. Parekh, A.K., Gallager, R.G.: A generalized processor sharing approach to flow control in integrated services networks: The single-node case. IEEE/ACM Transactions on Networking 1(3), 344–357 (1993)
27. Rizzo, G., Le Boudec, J.-Y.: Pay bursts only once does not hold for non-fifo guaranteed rate nodes. Performance Evaluation 62(1-4), 366–381 (2005)
28. Sariowan, H., Cruz, R.L., Polyzos, G.C.: Scheduling for quality of service guarantees via service curves. In: Proc. IEEE ICCCN, pp. 512–520 (September 1995)
29. Schmitt, J.B., Roedig, U.: Sensor network calculus - a framework for worst case analysis. In: Prasanna, V.K., Iyengar, S.S., Spirakis, P.G., Welsh, M. (eds.) DCOSS 2005. LNCS, vol. 3560, pp. 141–154. Springer, Heidelberg (2005)
30. Schmitt, J., Zdarsky, F., Fidler, M.: Delay bounds under arbitrary aggregate multiplexing: When network calculus leaves you in the lurch... In: Proc. IEEE INFOCOM (April 2008)
31. Skeie, T., Johannessen, S., Holmeide, O.: Timeliness of real-time IP communication in switched industrial ethernet networks. IEEE Transactions on Industrial Informatics 2(1), 25–39 (2006)

Flow Control with (Min,+) Algebra

Euriell Le Corronc, Bertrand Cottenceau, and Laurent Hardouin

Laboratoire d'Ingénierie des Systèmes Automatisés, Université d'Angers,
62, Avenue Notre Dame du Lac, 49000 Angers, France
{euriell.lecorronc,bertrand.cottenceau,laurent.hardouin}@univ-angers.fr
http://www.istia.univ-angers.fr/LISA/

Abstract. According to the theory of Network Calculus based on the (min,+) algebra, analysis and measure of worst-case performance in communication networks can be made easily. In this context, this paper deals with traffic regulation and performance guarantee of a network *i.e.* with flow control. At first, assuming that a minimum service provided by a network is known, we aim at finding the constraint over the input flow in order to respect a maximal delay or backlog. Then, we deal with the window flow control problem in the following manner: The data stream (from the source to the destination) and the acknowledgments stream (from the destination to the source) are assumed to be different and the service provided by the network is assumed to be known in an uncertain way, more precisely it is assumed to be in an interval. The results are obtained by considering the Residuation theory which allows functions defined over idempotent semiring to be inverted.

1 Introduction

Theory of (min,+) algebra enables the study of Discrete Event Dynamic Systems (DEDS) characterized by delay and synchronization phenomena such as production systems, communication networks and transportation systems (see [2]). Such systems can be described by linear models, thanks to the particular algebraic structure called *idempotent semiring* or *dioid*. In particular, the theory of Network Calculus aimed at worst-case performance analysis in communication networks. For instance, end-to-end delay or backlog can be computed with curves representing constraints over traffic and service provided by a network. Furthermore, operations defined over idempotent semiring and residuation theory allows some traffic control elements to be computed. Indeed, some model matching problems are already solved by the way of control structures (open-loop or close-loop structures) as presented in [7,9].

By leaning on Network Calculus as well as known control synthesis problem, the work introduced in this paper deals with control and performance guarantee of traffic in networks. On the one hand, the computation of the optimal constraint applied on the input flow in order to respect a maximum delay or backlog is given. By optimal we mean that it is the lower constraint such that the delay or backlog is satisfied: if the input flow is greater than this constraint, the resulting

T. Margaria and B. Steffen (Eds.): ISoLA 2010, Part I, LNCS 6415, pp. 184–197, 2010.

delay or backlog will be exceeded. This computation is made assuming that a minimum service provided by the network is known. On the other hand, optimal window size of a feedback configuration (window flow control) is studied. For that computation, a difference is made between the data stream (from the source to the destination) and the acknowledgments stream (from the destination to the source). Moreover, the service provided by the network is assumed to be included in an interval.

In order to introduce this work, the paper is organized as follows. Section 2 recalls the links between Network Calculus and (min,+) algebra. In particular some properties of the algebraic tools called *idempotent semiring* or *dioid* and classical operations of Network Calculus are presented. In the third section, the modelling of a communication network is given with cumulative functions, arrival and service curves and bounds on performances (delay and backlog) of a network. Finally, problems addressed previously are stated in the fourth section and an application is given in the last section.

2 An Algebraic Approach of Network Calculus

Network Calculus is a theory based on the (min,+) algebra and devoted to the analysis of performance guarantee in communication networks (see [5], [6] and [8]). This study lies on the particular algebraic structure called *idempotent semiring* whereas well-known operations as deconvolution and subadditive closure can be seen from the point of view of the residuation theory for the former and the solution of implicit equation $x = ax \oplus b$ for the latter. All these properties are recalled in this section.

2.1 (Min,+) Algebra

(Min,+) algebra is very closed to the lattice theory and the definition below of the *idempotent semiring* gives the basis of the particular algebraic structure used in this algebra (see [2]).

Definition 1. *An idempotent semiring $\mathcal{D}$ is a set endowed with two inner operations denoted $\oplus$ and $\otimes$. The sum $\oplus$ is associative, commutative, idempotent (i.e. $\forall a \in \mathcal{D}, a \oplus a = a$) and admits a neutral element denoted ε. The product $\otimes$ is associative, distributes over the sum and accepts e as neutral element.*

When $\otimes$ is commutative (*i.e.* $\forall a, b \in \mathcal{D}, a \otimes b = b \otimes a$), the idempotent semiring $\mathcal{D}$ is said to be commutative. Furthermore, an idempotent semiring is said to be complete if it is closed for infinite sums and if the product distributes over infinite sums. Then, the greatest element of $\mathcal{D}$ is denoted T (for *Top*) and represents the sum of all its elements ($T = \bigoplus_{x \in \mathcal{D}} x$).

Due to the idempotency of the addition, a canonical order relation can be associated with $\mathcal{D}$ by the following equivalences: $\forall a, b \in \mathcal{D}$, $a \succcurlyeq b \Leftrightarrow a = a \oplus b \Leftrightarrow b = a \wedge b$. Because of the lattice properties of a complete idempotent semiring, $a \oplus b$ is the least upper bound of $\mathcal{D}$ whereas $a \wedge b$ is its greatest lower bound.

An example of this structure is the idempotent semiring $\overline{\mathbb{R}}_{min}$ defined below.

Example 1 ((Min,+) algebra). The set $\overline{\mathbb{R}}_{min} = (\mathbb{R} \cup \{-\infty, +\infty\})$ endowed with the min operator as sum $\oplus$ and the pointwise addition as product $\otimes$ is a complete idempotent semiring where $\varepsilon = +\infty$, $e = 0$ and $T = -\infty$. On $\overline{\mathbb{R}}_{min}$, the greatest lower bound $\wedge$ takes the sense of the max operator.

Remark 1. It is important to note that the order relation in $\overline{\mathbb{R}}_{min}$ corresponds to the reverse of the natural order:

$$5 \oplus 3 = 3 \quad \Leftrightarrow \quad 3 \succcurlyeq 5 \quad \Leftrightarrow \quad 3 \leq 5.$$

In the rest of this document, the order relation of $\overline{\mathbb{R}}_{min}$ is used ($\succcurlyeq$ and $\preccurlyeq$) but the natural order (respectively $\leq$ and $\geq$) will be written clearly too when it will be necessary.

2.2 Other Algebraic Preliminaries

Residuation is a general notion in lattice theory which allows "pseudo-inverse" of some isotone maps (see [3] and [2]) to be defined. In particular, the residuation theory provides optimal solutions to inequalities $f(x) \preceq b$, where f is an order-preserving mapping (*i.e.* an isotone mapping: $a \preccurlyeq b \Rightarrow f(a) \preccurlyeq f(b)$) defined over ordered sets.

Definition 2 (Residuation). *Let $f : \mathcal{D} \to \mathcal{C}$ be an isotone mapping where $\mathcal{D}$ and $\mathcal{C}$ are complete idempotent semirings. Mapping f is said residuated if $\forall b \in \mathcal{C}$, the greatest element denoted $f^\sharp(b)$ of subset $\{x \in \mathcal{D} | f(x) \preccurlyeq b\}$ exists and belongs to this subset. Mapping $f^\sharp$ is called the residual of f. Furthermore, when f is residuated, $f^\sharp$ is the unique isotone mapping such that $f \circ f^\sharp \preccurlyeq \mathsf{Id}_\mathcal{C}$ and $f^\sharp \circ f \succcurlyeq \mathsf{Id}_\mathcal{D}$, where $\mathsf{Id}_\mathcal{C}$ and $\mathsf{Id}_\mathcal{D}$ are respectively the identity mappings on $\mathcal{C}$ and $\mathcal{D}$.*

Fixed point theory allows one to find greatest finite solutions to equations $f(x) = x$, where f is an isotone mapping defined over complete idempotent semiring $\mathcal{D}$. In particular, thanks to the following theorem, the optimal solution of the implicit equation $x = ax \oplus b$ is provided.

Theorem 1. *[2, section 4.5.3] Implicit equation $x = ax \oplus b$ defined over a complete dioid $\mathcal{D}$ admits $x = a^\star b$ as lower solution where $\forall a \in \mathcal{D}, a^\star = \bigoplus_{i \geq 0} a^i$ and $a^0 = e$.*

These two theories will be necessary in the definition of operations linked to Network Calculus, as the next section shows it.

2.3 Operations of Network Calculus

Once (min,+) algebra and other tools are defined, first main operations used by Network Calculus as pointwise minimum and inf-convolution can be given. To this end, the set $\mathcal{F}$ brings together non-decreasing functions $f : \mathbb{R} \mapsto \overline{\mathbb{R}}_{min}$ where $f(t) = 0$ for $t < 0$. A restriction of this set is the set $\mathcal{F}_0$ where $f(0) = 0$. Let now f and g be two functions of $\mathcal{F}_0$, the following operations are defined:

- pointwise minimum

$$(f \oplus g)(t) = \min[f(t), g(t)],$$

- pointwise maximum

$$(f \wedge g)(t) = \max[f(t), g(t)],$$

- inf-convolution

$$(f * g)(t) \triangleq \bigoplus_{\tau \geq 0} \{f(\tau) \otimes g(t-\tau)\} = \min_{\tau \geq 0} \{f(\tau) + g(t-\tau)\}.$$

Thanks to these operations, another idempotent semiring can be defined.

Definition 3. *The set $\mathcal{F}_0$ endowed with the two inner operations $\oplus$ as pointwise minimum and $*$ as inf-convolution is a commutative idempotent semiring denoted $\{\mathcal{F}_0, \oplus, *\}$ where ε and e are defined by:*

$$\forall t, \varepsilon : t \mapsto +\infty \quad \textit{and} \quad e : t \mapsto \begin{cases} 0 & \textit{for } t < 0, \\ +\infty & \textit{for } t \geq 0. \end{cases}$$

Remark 2. As in usual algebra, operator $*$ can be omitted in order to save place:

$$ab = a * b.$$

Then, two another well-known operations of Network Calculus are the one of *deconvolution* denoted $\oslash$ and the one of *subadditive closure* denoted $\star$. Firstly, thanks to the residuation theory (see Definition 2), mapping $R_a : x \mapsto x * a$ defined over $\mathcal{F}_0$ is said to be residuated. Its residual is usually denoted $R_a^\sharp : x \mapsto x \oslash a$ and called *deconvolution*. Therefore, $b \oslash a$ is the greatest solution to inequality $x * a \preccurlyeq b$, *i.e.*:

$$b \oslash a = \hat{x} = \bigoplus \{x \mid x * a \preccurlyeq b\}.$$

Remark 3. This operation of deconvolution is also called *right quotient* and a similar mapping, called *left quotient* and denoted $L_a^\sharp : x \mapsto a \backslash\!\!\backslash x$ exists. This mapping is the residual of $L_a : x \mapsto a * x$ defined over $\mathcal{F}_0$ and $a \backslash\!\!\backslash b = \hat{x} = \bigoplus \{x \mid a * x \preccurlyeq b\}$. However, since $\mathcal{F}_0$ is commutative $b \oslash a = a \backslash\!\!\backslash b$.

Secondly, according to theorem 1, the subadditive closure operation $\star$ takes the sense of the optimal solution of a given implicit equation $a^\star = \bigwedge \{x \mid x = ax \oplus e\}$. Finally, $\forall f, g \in \mathcal{F}_0$ operations of deconvolution and subadditive closure are given:

- deconvolution

$$(f \oslash g)(t) \triangleq \bigwedge_{\tau \geq 0} \{f(\tau) - g(t-\tau)\} = \max_{\tau \geq 0} \{f(\tau) - g(\tau - t)\},$$

- subadditive closure

$$f^\star(t) \triangleq \bigoplus_{\tau \geq 0} f^\tau(t) = \min_{\tau \geq 0} f^\tau(t) \quad \text{with} \quad f^0(t) = e.$$

Numerous properties are associated with both deconvolution and subadditive closure. The following theorem brings together some of useful properties.

Theorem 2. *Firstly,* $\forall x, y, a \in \{\mathcal{F}_0, \oplus, *\}$*:*

$$x \preccurlyeq y \Rightarrow \begin{cases} a \backslash x \preccurlyeq a \backslash y & (x \mapsto a \backslash x \text{ is isotone}), \\ x \backslash a \succcurlyeq y \backslash a & (x \mapsto x \backslash a \text{ is antitone}). \end{cases} \tag{1}$$

Then:

$$(x * a) \oslash a \succcurlyeq x, \tag{2}$$

$$x \oslash (b * a) = (x \oslash a) \oslash b, \tag{3}$$

$$a \oslash b \succcurlyeq x \Leftrightarrow a \succcurlyeq xb, \tag{4}$$

$$(a^\star)^\star = a^\star. \tag{5}$$

And in particular about the subadditive closure:

$$a^\star = \bigoplus \{x \mid x^\star \preccurlyeq a^\star\}, \tag{6}$$

$$a^\star = \bigwedge \{x \mid x = x^\star,\ x \succcurlyeq a\}. \tag{7}$$

Proof. Proofs of these equations are found in literature. For equation (1), see [10] whereas for equations (2) until (5) see [2]. Finally, for equations (6) and (7) see [9] with the following precision for the latter: $a^\star$ is a solution of this equation because $a^\star \succcurlyeq a$ and it is also the smallest one because $(a^\star)^\star \succcurlyeq a^\star \Leftrightarrow a^\star \succcurlyeq a^\star \Leftrightarrow x^\star \succcurlyeq a^\star$. □

3 Network Calculus Modelling

3.1 Input and Output Flows, Arrival and Service Curves

Input and output flows. A communication network can be seen as a blackbox denoted S with an input flow and an output flow. These flows are respectively described by cumulative functions belonging to $\mathcal{F}_0$ and denoted u and y. Element $u(t)$ corresponds to the total amount of data introduced in the system until time t whereas $y(t)$ corresponds to the total amount of data that has left the system until this time. The main assumption made about input and output flows is a characteristic of causality:

$$u \preccurlyeq y,$$

which means that for all t, $u(t) \geq y(t)$. So, the amount of data leaving the network is always lower than the one getting in.

In order to guarantee performance in network, constraints are applied over these flows. For instance, an arrival curve is applied over the input flow whereas the service provided by S is constrained by a lower curve as well as an upper curve.

Arrival curve. One says that a given flow $u \in \mathcal{F}_0$ is constrained by an arrival curve $\alpha \in \mathcal{F}_0$ if it is such that $\forall s \leq t \in \mathbb{R}_+$, $u(t) - u(s) \leq \alpha(t-s)$ (u is said α-smooth). So, the amount of data arriving between time s and time t is at most $\alpha(t-s)$. Firstly, according to the definition of the inf-convolution:

$$u(t) - u(s) \leq \alpha(t-s) \quad \Leftrightarrow \quad u \leq \alpha u. \quad \Leftrightarrow \quad u \succcurlyeq \alpha u.$$

Secondly, thanks to the isotony of the inf-convolution, this inequality can be written as below:

$$u \succcurlyeq \alpha u \;\Rightarrow\; \alpha u \succcurlyeq (\alpha^2 u) \;\Rightarrow\; (\alpha^2 u) \succcurlyeq (\alpha^3 u) \;\Rightarrow\; \ldots$$

and therefore:

$$u = u \oplus (\alpha u) \oplus (\alpha^2 u) \oplus \ldots = \bigoplus_{n \geq 0} \alpha^n u.$$

Finally:

$$u \succcurlyeq \alpha u \quad \Leftrightarrow \quad u = \alpha^\star u. \tag{8}$$

So, α is an arrival curve[1] for u if and only if for the input flows considered we have: $u = \alpha^\star u$.

Service curve. As regards to the service provided by S, it is framed by two service curves $\underline{\beta}$ and $\overline{\beta} \in \mathcal{F}_0$ such that $\underline{\beta} \preccurlyeq \overline{\beta}$ ($\underline{\beta} \geq \overline{\beta}$). These curves constitute interval $[\,\underline{\beta}\,,\,\overline{\beta}\,]$ where $\overline{\beta}$ corresponds to the minimum service provided by S for all input flows and $\underline{\beta}$ corresponds to its maximum service. Then, output flow y is included in an interval too:

$$\underline{\beta} u \preccurlyeq y \preccurlyeq \overline{\beta} u \quad \Leftrightarrow \quad y \in [\,\underline{\beta} u\,,\,\overline{\beta} u\,]. \tag{9}$$

All these Network Calculus elements (input and output flows, arrival and service curve) are illustrated in Figure 1.

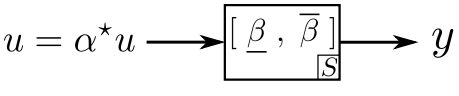


Fig. 1. Network Calculus diagram

3.2 Performance Characteristics: Delay and Backlog

Two characteristics used as performance indicator in Network Calculus are the delay and the backlog (see [4]). The former denoted $d(t)$ corresponds to the waiting time of a paquet in a FIFO order whereas the latter denoted $b(t)$ is the amount of paquets in a network at time t. Let u and y be the input and the output flow of a network:

$$d(t) \triangleq \inf_{\tau \geq 0} \{\tau \mid u(t) \leq y(t+\tau)\},$$
$$b(t) \triangleq u(t) - y(t).$$

[1] And $\alpha^\star$ is also an arrival curve for u.

These data are given for all time t in the network. However, according to the following theorem coming from the second order theory of (min,+) linear systems detailed in [10] and used in [11], upper bounds on their worst-case can be measured easily.

Theorem 3. *Let v_1 and v_2 be two functions of $\mathcal{F}$ where*[2] *$v_1 \preccurlyeq v_2$. Function $v_1 \not\phi v_2$ is called the correlation of v_1 over v_2 and contains the maximal distances, denoted $\tau_{\max}$ and $\nu_{\max}$, between v_1 and v_2 respectively in time and event domain. More precisely, $\tau_{\max}$ and $\nu_{\max}$ are such that:*

$$\tau_{\max} = \inf_{\tau \geq 0} \{\tau \mid (v_1 \not\phi v_2)(-\tau) \leq 0\},$$
$$\nu_{\max} = (v_1 \not\phi v_2)(0).$$

Remark 4. It is possible that $v_1 \not\phi v_2 = \varepsilon$. In such a case, maximal time and event distances $\tau_{\max}$ and $\nu_{\max}$ are infinite.

Then, thanks to theorem 3, we are able to provide two kinds of distances for a network S:

- if input and output flows u and y of S are assumed to be known then, its *maximal delay and backlog* can be computed,
- if arrival curve of u and minimum service curve of S are assumed to be known then, *upper bounds* on maximal delay and backlog can be computed.

These measures as well as links between them are given in the following proposition. This is a different formulation of some well known results (see [8, §3.1.11]) but with different tools.

Proposition 1. *On the one hand, let $u, y \in \mathcal{F}_0$ be input and output flows of a network S such that $u \preccurlyeq y$. On the other hand, let α be the arrival curve of input u such that $u = \alpha^\star u$ and $\overline{\beta}$ be a minimum service curve of S such that $y \preccurlyeq \overline{\beta} u$. Then:*

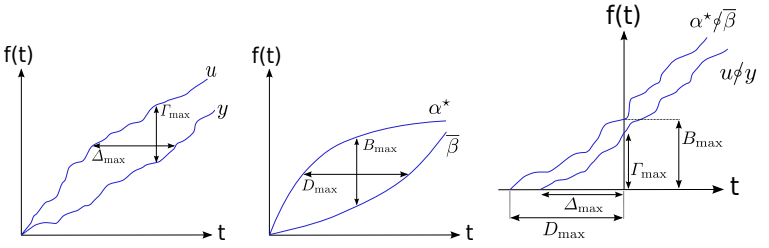
$$\begin{array}{ccccc} d(t) & \leq & \Delta_{\max} = \inf\limits_{\Delta \geq 0} \{(u \not\phi y)(-\Delta) \leq 0\} & \leq & D_{\max} = \inf\limits_{D \geq 0} \{(\alpha^\star \not\phi \overline{\beta})(-D) \leq 0\}, \\ b(t) & \leq & \Gamma_{\max} = (u \not\phi y)(0) & \leq & B_{\max} = (\alpha^\star \not\phi \overline{\beta})(0). \end{array}$$

Distances $\Delta_{\max}$ and $\Gamma_{\max}$ are the maximal delay and backlog of S whereas $D_{\max}$ and $B_{\max}$ are their upper bounds (see Figure 2).

Proof. Since $u = \alpha^\star u$ and $y \preccurlyeq \overline{\beta} u$ (see equations (8) and (9)) are the relation between real flows of network S and constraints over these flows, the following inequality shows that from the correlation $u \not\phi y$ another correlation with $\alpha^\star$ and $\overline{\beta}$ is found:

$$\begin{array}{lcll} y \preccurlyeq \overline{\beta} u & \Rightarrow & u \not\phi y \succcurlyeq (\alpha^\star u) \not\phi (\overline{\beta} u) & \text{since } u = \alpha^\star u \text{ and see (1)}, \\ & & \succcurlyeq ((\alpha^\star u) \not\phi u) \not\phi (\overline{\beta}) & \text{see (3)}, \\ & & \succcurlyeq \alpha^\star \not\phi \overline{\beta} & \text{see (2)}. \end{array}$$

[2] Recall that $v_1 \preccurlyeq v_2 \Leftrightarrow \forall t, v_1(t) \geq v_2(t)$.



(a) Flows u and y with distances $\Delta_{\max}$ and $\Gamma_{\max}$

(b) Curves $\alpha^\star$ and $\overline{\beta}$ with distances $D_{\max}$ and $B_{\max}$

(c) Correlations $u \not\phi y$ and $\alpha^\star \not\phi \beta$

Fig. 2. Maximal delay and backlog and their upper bounds

So, according to the $\overline{\mathbb{R}}_{min}$ order relation:

$$u \not\phi y \succcurlyeq \alpha^\star \not\phi \overline{\beta} \quad \Leftrightarrow \quad u \not\phi y \leq \alpha^\star \not\phi \overline{\beta}. \qquad \square$$

Remark 5. In the Network Calculus literature, maximal distances $D_{\max}$ and $B_{\max}$ are obtained by horizontal and vertical deviations between elements of correlation $\alpha^\star \not\phi \overline{\beta}$ as shown in Figure 2b.

3.3 Functions Associated to Delay and Backlog

In the next sections dealing with some control problems, we will need to handle given delay and backlog as fixed value of pure delay and amount of data. To this end, particular functions are defined below.

Definition 4. *Let τ and ν be respectively a pure delay and an amount of data. Then, function denoted δ_τ is defined by:*

$$\delta_\tau(t) = \begin{cases} 0 & \text{for } t < \tau, \\ +\infty & \text{for } t \geq \tau, \end{cases}$$

and the one denoted γ_ν by:

$$\gamma_\nu(t) = \begin{cases} \nu & \text{for } t < 0, \\ +\infty & \text{for } t \geq 0. \end{cases}$$

Remark 6. Some properties can be associated to these functions:

$$\delta_\tau * \delta_{-\tau} = e,$$
$$\gamma_\nu * \gamma_{-\nu} = e.$$

Moreover, in relation to input flow u, these functions are such that:

$$\forall t, \ (\delta_\tau * u)(t) = u(t - \tau),$$
$$\forall t, \ (\gamma_\nu * u)(t) = \nu + u(t).$$

4 Flow Control

In this section, we consider the traffic regulation in order to get a guaranteed performance of a network, this is known as the flow control.

4.1 Arrival Curve Computation

The first problem addressed in this paper is the next one. Assuming that a minimum service provided by a network is known, we aim at finding the arrival curve, *i.e.* a constraint applied over input flow, in order to respect a maximal delay or backlog. By definition of an arrival curve (see equation (8) with $u \succcurlyeq \alpha u$ and so $u = \alpha^\star u$), this optimal curve is a subadditive closure. Moreover, an optimal curve represents the minimal constraint applied over the input in order to eventually reach but not exceed the given delay or backlog. The problem from the point of view of time performance is given in the following proposition.

Proposition 2. *Let $\overline{\beta}$ be a minimal service curve of a network S and τ be a fixed worst end-to-end delay. The optimal arrival curve $\hat{\alpha}^\star$ which guarantees the respect of τ is given by:*

$$\hat{\alpha}^\star = \bigwedge\{\alpha^\star \mid \alpha^\star \succcurlyeq \delta_{-\tau}\overline{\beta}\} = (\delta_{-\tau}\overline{\beta})^\star$$

where $\delta_{-\tau}$ is the function associated with τ.

Proof. First of all, according to proposition 1 and definition 4, upper bound $D_{\max}$ of worst end-to-end delay is given by correlation $\alpha^\star \not\phi \overline{\beta}$ and can be represented by function $\delta_{-D_{\max}}$. So, the following relation is given:

$$D_{\max} = -\sup\{D \mid (\alpha^\star \not\phi \overline{\beta})(D) \leq 0\} \quad \Rightarrow \quad \alpha^\star \not\phi \overline{\beta} \succcurlyeq \delta_{-D_{\max}}.$$

Then, if the worst end-to-end delay τ is chosen $D_{\max} = \tau \Leftrightarrow \delta_{-D_{\max}} = \delta_{-\tau}$ and thanks to equation (4), arrival curve $\alpha^\star$ has to follow these following inequalities:

$$\alpha^\star \not\phi \overline{\beta} \succcurlyeq \delta_{-\tau} \quad \Leftrightarrow \quad \alpha^\star \succcurlyeq \delta_{-\tau}\overline{\beta}.$$

Finally, thanks to equation (7), the minimal $\alpha^\star$ which respects the inequality is $(\delta_{-\tau}\overline{\beta})^\star$. □

The next proposition states the problem from the point of view of data performance.

Proposition 3. *Let $\overline{\beta}$ be the known minimal service curve of a network S and ν be a fixed worst backlog. The optimal arrival curve $\hat{\alpha}^\star$ which guarantees the respect of ν is given by:*

$$\hat{\alpha}^\star = \bigwedge\{\alpha^\star \mid \alpha^\star \succcurlyeq \gamma_{\nu}\overline{\beta}\} = (\gamma_{\nu}\overline{\beta})^\star$$

where γ_ν is the function associated with ν.

Proof. The proof is the same as in proposition 2. □

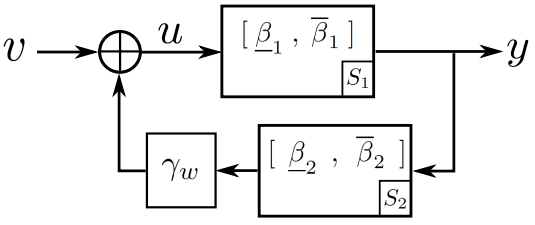


Fig. 3. Chosen configuration of the window flow control system

4.2 Window Flow Control

The second problem of traffic regulation and performance guarantee is the one of the window flow control where its optimal window size is computed.

First of all, let us recall this control context. A window flow controller aims at bounding the amount of data admitted in a network in such a way that its total amount in transit is always less than some positive number, *i.e.* the window size. This problem has already been treated in literature but not in the same manner. The window flow control introduced in [8] do not have the same model than in [6]. In this paper, we adopt the Chang's modelling which is homogeneous with the one introduced in [7].

Moreover, this problem is studied here with two associated configurations. On the one hand, the service provided by the network is assumed to be included in interval. Indeed, assuming that minimum and maximum service curves are known, the size of the window can be computed as well as for the worst case than for the best case of traffic without damaging the service provided. On the other hand, a difference is made between the data stream (from the source to the destination) and the acknowledgments stream (from the destination to the source) since these acknowledgments requires considerably less bandwidth than the data itself (see [1]). So, the computation of the window size will have benefit of this profit of bandwidth. This configuration is described in the following proposition and illustrated in Figure 3.

Proposition 4. *Let S_1 be the system representing the data stream where $[\ \underline{\beta}_1\ ,\ \overline{\beta}_1\ ]$ ($\underline{\beta}_1 \preccurlyeq \overline{\beta}_1$) is the interval containing its provided service. In the same way, let S_2 be the system representing the acknowledgments stream where $[\ \underline{\beta}_2\ ,\ \overline{\beta}_2\ ]$ ($\underline{\beta}_2 \preccurlyeq \overline{\beta}_2$) is the interval containing its provided service. Then, let γ_w be the representative function of the window size w. The service curve of the whole system is included in the interval:*

$$[\ \underline{\beta}_1(\gamma_w\underline{\beta}_2\underline{\beta}_1)^\star\ ,\ \overline{\beta}_1(\gamma_w\overline{\beta}_2\overline{\beta}_1)^\star\ ].$$

Proof. The output flow y is described by the following equation:

$$\underline{\beta}_1 u \preccurlyeq\ y\ \preccurlyeq \overline{\beta}_1 u,$$

whereas intermediate flow u is included in:

$$\begin{aligned} \min(v, \gamma_w \underline{\beta}_2 \underline{\beta}_1 u) \preccurlyeq\ & u \preccurlyeq \min(v, \gamma_w \overline{\beta}_2 \overline{\beta}_1 u), \\ v \oplus \gamma_w \underline{\beta}_2 \underline{\beta}_1 u \preccurlyeq\ & u \preccurlyeq v \oplus \gamma_w \overline{\beta}_2 \overline{\beta}_1 u && \text{see operator } \oplus, \\ (\gamma_w \underline{\beta}_2 \underline{\beta}_1)^\star v \preccurlyeq\ & u \preccurlyeq (\gamma_w \overline{\beta}_2 \overline{\beta}_1)^\star v && \text{see theorem 1.} \end{aligned}$$

Therefore:

$$\underline{\beta}_1 (\gamma_w \underline{\beta}_2 \underline{\beta}_1)^\star v \preccurlyeq\ y\ \preccurlyeq \overline{\beta}_1 (\gamma_w \overline{\beta}_2 \overline{\beta}_1)^\star v. \qquad \square$$

By considering this configuration, the computation of the optimal window size $\hat{w}$ can be studied. The chosen point of view is to compute a minimal window size such that the global network behavior, *i.e.* the controlled one, is the same as the open-loop network behavior, *i.e.* the one of S_1 only. The behavior of S_1 is described by the interval of service curve $[\ \underline{\beta}_1\ ,\ \overline{\beta}_1\ ]$, this objective can be stated as follows:

$$\hat{\gamma_w} \ = \ \bigoplus \{\gamma_w \mid \underline{\beta}_1 (\gamma_w \underline{\beta}_2 \underline{\beta}_1)^\star = \underline{\beta}_1 \quad \text{and} \quad \overline{\beta}_1 (\gamma_w \overline{\beta}_2 \overline{\beta}_1)^\star = \overline{\beta}_1 \}. \tag{10}$$

The following proposition puts forward the computation of such a window size.

Proposition 5. *In order to obtain a behavior of the closed-loop system unchanged in comparison to the one of the open-loop (see equation (10)), the optimal window size $\hat{w}$ represented by function $\hat{\gamma_w}$ is given below:*

$$\hat{\gamma_w} = (\underline{\beta}_1 \oslash \underline{\beta}_1 \not\oslash (\underline{\beta}_2 \underline{\beta}_1)) \wedge (\overline{\beta}_1 \oslash \overline{\beta}_1 \not\oslash (\overline{\beta}_2 \overline{\beta}_1)).$$

Proof. Firstly, by considering the minimal bound, let $\underline{G}_c$ be the minimal behavior of the controlled network and $\underline{G}_{ref}$ be the reference behavior we want to reach, so the one of $\underline{\beta}_1$:

$$\underline{G}_c = \underline{\beta}_1 (\gamma_w \underline{\beta}_2 \underline{\beta}_1)^\star \quad \text{and} \quad \underline{G}_{ref} = \underline{\beta}_1.$$

Equation (10) can be written as follow:

$$\hat{\gamma_w} \ = \ \bigoplus \{\gamma_w \mid \underline{G}_c \preccurlyeq \underline{G}_{ref}\}.$$

Then:

$$\begin{aligned} \underline{G}_c \preccurlyeq \underline{G}_{ref} \quad & \Leftrightarrow \quad \underline{\beta}_1 (\gamma_w \underline{\beta}_2 \underline{\beta}_1)^\star \preccurlyeq \underline{\beta}_1, \\ & \Leftrightarrow \quad (\gamma_w \underline{\beta}_2 \underline{\beta}_1)^\star \preccurlyeq \underline{\beta}_1 \oslash \underline{\beta}_1 && \text{see (4)}, \\ & \Leftrightarrow \quad \gamma_w \underline{\beta}_2 \underline{\beta}_1 \preccurlyeq \underline{\beta}_1 \oslash \underline{\beta}_1 && \text{since } a \preccurlyeq a^\star, \\ & \Leftrightarrow \quad \gamma_w \preccurlyeq \underline{\beta}_1 \oslash \underline{\beta}_1 \not\oslash (\underline{\beta}_2 \underline{\beta}_1) && \text{see (4)}. \end{aligned} \tag{11}$$

Secondly, this proof is the same for the maximal bound and thus we obtain:

$$\gamma_w \preccurlyeq \overline{\beta}_1 \oslash \overline{\beta}_1 \not\oslash (\overline{\beta}_2 \overline{\beta}_1). \tag{12}$$

Then, in order to satisfy both equations (11) and (12), function γ_w is given by:

$$\hat{\gamma_w} = (\underline{\beta}_1 \oslash \underline{\beta}_1 \not\oslash (\underline{\beta}_2 \underline{\beta}_1)) \wedge (\overline{\beta}_1 \oslash \overline{\beta}_1 \not\oslash (\overline{\beta}_2 \overline{\beta}_1)). \qquad \square$$

Remark 7. Thanks to this optimal window size, the interval including the service curve of the controlled system is the same as the one of the open-loop system:

$$[\ \underline{\beta}_1(\hat{\gamma_w}\underline{\beta}_2\underline{\beta}_1)^\star\ ,\ \overline{\beta}_1(\hat{\gamma_w}\overline{\beta}_2\overline{\beta}_1)^\star\ ] \quad = \quad [\ \underline{\beta}_1\ ,\ \overline{\beta}_1\ ].$$

5 Application: Window Flow Control with a Given Delay

Let us see an example of a window flow control with a given delay to respect. This application takes the main propositions of this paper into account, namely proposition 2 about the computation of an arrival curve according to a given delay (its backlog version given in proposition 3 is not treated in this example) and proposition 5 about optimal size of a window flow controller.

5.1 Configuration

For this application, the scheme of the network is the same as described in proposition 4 and illustrated in Figure 3. All the service provided by element S_1 is included in interval $[\ \underline{\beta}_1\ \ \overline{\beta}_1\ ]$. Services curves $\underline{\beta}_1$ and $\overline{\beta}_1$ are rate-latency functions with a latency of $16ms$ for the former, $20ms$ for the latter and a rate of $100Mb/s$ for both of them:

$$\underline{\beta}_1(t) = 16ms + 100Mb/s \cdot t \quad \text{and} \quad \overline{\beta}_1(t) = 20ms + 100Mb/s \cdot t.$$

By considering the service provided by element S_2 included in interval $[\ \underline{\beta}_2\ \ \overline{\beta}_2\ ]$ with rate-latency functions as services curves $\underline{\beta}_2$ and $\overline{\beta}_2$:

$$\underline{\beta}_2(t) = 12ms + 100Mb/s \cdot t \quad \text{and} \quad \overline{\beta}_2(t) = 14ms + 100Mb/s \cdot t.$$

For this network, the upper bound $D_{\max}$ of the worst end-to-end delay from v to y is fixed to $90ms$. This delay is represented by function $\delta_{-D_{\max}}$.

5.2 Computation of the Arrival Curve $\hat{\alpha}^\star$

Firstly, proposition 2 is applied in order to find the minimal arrival curve $\hat{\alpha}^\star$ which allows $D_{\max}$ to be respected in the open-loop context. Thus:

$$\begin{aligned} \hat{\alpha}^\star = (\delta_{-D_{\max}}\overline{\beta}_1)^\star \quad \Rightarrow \quad \hat{\alpha}^\star(t) &= (-90ms) * (20ms + 100Mb/s \cdot t), \\ &= 9Mb + 100Mb/s \cdot t. \end{aligned}$$

This arrival curve is the lowest one, so the less restrictive one enabling eventually to reach but not to exceed the given delay. If an arrival curve still less restrictive is chosen, the network will be subjected to congestions and the maximum end-to-end delay of the network will increase.

5.3 Computation of the Window Size $\hat{w}$

Secondly, we can compute the optimal window size $\hat{w}$ for this configuration. However, the maximum end-to-end delay $D_{\max}$ has to be respected again and thus the optimal arrival curve $\hat{\alpha}^\star$ previously computed is used as follows: input v of the global system is constrained by $\hat{\alpha}^\star$ such that $v = \hat{\alpha}^\star v$. So, the open-loop behavior is the following interval:

$$\hat{\alpha}^\star[\ \underline{\beta}_1\ ,\ \overline{\beta}_1\ ]$$

whereas the closed-loop one is:

$$\hat{\alpha}^\star[\ \underline{\beta}_1(\hat{\gamma_w}\underline{\beta}_2\underline{\beta}_1)^\star\ ,\ \overline{\beta}_1(\hat{\gamma_w}\overline{\beta}_2\overline{\beta}_1)^\star\ ].$$

Then, proposition 5 is applied in order to find the minimal window size $\hat{w}$ which allows the same behavior in closed-loop context than in open-loop context to be obtained and function $\hat{\gamma_w}$ is given by:

$$\hat{\gamma_w} = ((\underline{\beta}_1\hat{\alpha}^\star)\backslash(\underline{\beta}_1\hat{\alpha}^\star)\not/(\underline{\beta}_2\underline{\beta}_1)) \wedge ((\overline{\beta}_1\hat{\alpha}^\star)\backslash(\overline{\beta}_1\hat{\alpha}^\star)\not/(\overline{\beta}_2\overline{\beta}_1)).$$

The proof of this result is left to reader by following the one of proposition 5. Finally, the optimal window size is obtained:

$$\hat{w} = 2,8Mb.$$

This window size is the minimal one for the largest bandwidth of the network without congestion it. Moreover, this window size respect the maximum end-to-end delay given in the assumption.

6 Conclusion

In this paper, traffic regulation and performance guarantee of a network have been treated. First of all, we recalled the algebraic linked to Network Calculus operations thanks to the (min,+) algebra and the residuation theory. Once these useful properties defined, we used them in the context of flow control in order to solve two problems enabling to avoid congestion in the network.

The first case shows the computation of an optimal arrival curve in order to respect a maximal delay or backlog, assuming that the minimum service provided by a network is known. This arrival curve is said to be optimal because it is the less restrictive one where the given delay is not exceeded.

The second case brings forward the computation of a window size in a closed-loop structure. Assuming that the data stream and the acknowledgments are different, this window size is said to be optimal. Moreover, the service provided by network elements are included in an interval so the window flow controller is computed as well as for the worst case than for the best case of traffic.

Finally, an example applies propositions made in order to solve these two problems and optimal arrival curve and window size are found.

References

1. Agrawal, R., Cruz, R.L., Okino, C., Rajan, R.: Performance bounds for flow control protocols. IEEE/ACM Transactions on Networking (TON) 7(3), 310–323 (1999)
2. Baccelli, F., Cohen, G., Olsder, G.J., Quadrat, J.-P.: Synchronisation and linearity: an algebra for discrete event systems. Wiley and sons, Chichester (1992)
3. Blyth, T.S., Janowitz, M.F.: Residuation theory. Pergamon, Oxford (1972)
4. Bouillard, A., Jouhet, L., Thierry, E.: Computation of a (min,+) multi-dimensional convolution for end-to-end performance analysis. In: Proceedings of the 3rd International Conference on Performance Evaluation Methodologies and Tools, ValueTools 2008 (2008)
5. Chang, C.S.: On deterministic traffic regulation and service guarantees: asystematic approach by filtering. IEEE Transactions on Information Theory 44(3), 1097–1110 (1998)
6. Chang, C.S.: Performance guarantees in communication networks. Springer, Heidelberg (2000)
7. Cottenceau, B., Hardouin, L., Boimond, J.-L., Ferrier, J.-L.: Model reference control for timed event graphs in dioids. Automatica 37(9), 1451–1458 (2001)
8. Le Boudec, J.-Y., Thiran, P.: Network calculus: a theory of deterministic queuing systems for the internet. Springer, Heidelberg (2001)
9. Maia, C.A., Hardouin, L., Santos-Mendes, R., Cottenceau, B.: Optimal closed-loop control of timed event graphs in dioids. IEEE Transactions on Automatic Control 48(12), 2284–2287 (2003)
10. Plus, M.: Second order theory of min-linear systems and its application to discrete event systems. In: Proceedings of the 30th IEEE Conference on Decision and Control, Brighton, CDC 1991 (1991)
11. Santos-Mendes, R., Cottenceau, B., Hardouin, L.: Adaptive feedback control for (max,+)-linear systems. In: Proceedings of the 10th IEEE Conference on Emerging Technologies and Factory Automation, ETFA 2005 (2005)

An Interface Algebra for Estimating Worst-Case Traversal Times in Component Networks*

Nikolay Stoimenov[1], Samarjit Chakraborty[2], and Lothar Thiele[1]

[1] Computer Engineering and Networks Laboratory, ETH Zurich, Switzerland
{stoimenov,thiele}@tik.ee.ethz.ch
[2] Institute for Real-Time Computer Systems, TU Munich, Germany
samarjit@tum.de

Abstract. Interface-based design relies on the idea that different components of a system may be developed independently and a system designer can connect them together only if their interfaces match, without knowing the details of their internals. In this paper we propose an interface algebra for analyzing networks of embedded systems components. The goal is to be able to compute worst-case traversal times and verify their compliance to provided deadlines in such component networks in an incremental manner, i.e., as and when new components are added or removed from the network. We lay the basic groundwork for this algebra and show its utility through an illustrative example.

1 Introduction

Today most embedded systems consist of a collection of computation and communication components that are supplied by different vendors and assembled by a system manufacturer. Such a component-based design methodology is followed in several domains such as automotive, avionics, and consumer electronics. The system manufacturer responsible for component assembly has to take design decisions (related to the choice of components and how they are to be connected, e.g., using a bus or a network-on-chip) and perform system analysis. One such important analysis is related to the computation of end-to-end delays or worst-case traversal times (WCTTs) of data through the component network.

To carry out such an analysis, it is important to model the internals of each component, which is often difficult because of the complexity of the components and their proprietary nature. To get around this problem, an *interface*-based design methodology may be appropriate. Here, components may be connected if and only if their interfaces *match*. Further, the timing behavior of the network may be computed in an incremental manner – i.e., as and when new components are added or existing ones are modified – without always having to analyze the

* The work is partially supported by NCCR-MICS, a center supported by the Swiss National Science Foundation under grant number 5005-67322, and by the DFG through the SFB/TR28 "Cognitive Automobiles".

T. Margaria and B. Steffen (Eds.): ISoLA 2010, Part I, LNCS 6415, pp. 198–213, 2010.

entire system from scratch. In this paper, we develop an *interface algebra* for estimating the worst-case traversal times and verify their compliance to provided upper bounds, where the different components exchange data through first-in first-out (FIFO) buffers. Such architectures are common for streaming applications, e.g., audio/video processing and distributed controllers where data flows from sensors to actuators while getting processed on multiple processors. Our proposed interface algebra is an extension of [4,8] and is motivated by the concept of *assume/guarantee* interfaces from [1]. In particular, we cast the *Real-Time Calculus* framework from [3] within the assume/guarantee interface setting. At a high level, two interfaces *match* when the guarantees associated with one of them comply with the assumptions associated with the other. Our main technical result is Theorem 1 which describes how to compute the *assumptions* and *guarantees* associated with components based on a *monotonicity* property. This result is then used to compute the WCTT in a component network incrementally, and validate that it complies to a given deadline. Our approach may be summarized in the following three steps:

1. Define an abstract component that describes the real-time properties of a concrete system component. This involves defining proper abstractions for component inputs and outputs, and internal component relations that meaningfully relate abstract inputs to abstract outputs.
2. To derive the interface of an abstract component, we need to define interface variables as well as input and output predicates on these interface variables.
3. Derive the internal interface relations that relate incoming guarantees and assumptions to outgoing guarantees and assumptions of a component's interfaces.

In the setting we study here, event streams are processed on a sequence of components. An event or data stream described by the cumulative function $R(t)$ enters the input buffer of the component and is processed by the component whose availability is described by the cumulative function $C(t)$. Formally, the cumulative functions $R(t) \in \mathbb{R}^{\geq 0}$ and $C(t) \in \mathbb{R}^{\geq 0}$ for $t \geq 0$ denote the number of events/data items that have been received or could be processed within the time interval $[0, t)$, respectively. After being processed, events are emitted on the component's output, resulting in an outgoing event stream $R'(t)$. The remaining resources that were not consumed are available for use and are described by an outgoing resource availability trace $C'(t)$. The relations between $R(t)$, $C(t)$, $R'(t)$ and $C'(t)$ depend on the component's processing semantics. For example, Greedy Processing (GP) denotes that events are always processed when there are resources available. Typically, the outgoing event stream $R'(t)$ will not equal the incoming event stream $R(t)$ as it may, for example, exhibit more or less jitter.

While cumulative functions such as $R(t)$ or $C(t)$ describe one concrete trace of an event stream or a resource availability, *variability characterization curves* (VCCs) capture all possible traces using upper and lower bounds on their timing

properties. The *arrival* and *service curves* from Network Calculus [5] are specific instances of VCCs and are more expressive than traditional event stream models such the periodic, periodic with jitter, sporadic, etc. Arrival curves $\alpha^l(\Delta)$ and $\alpha^u(\Delta)$ denote the minimum and the maximum number of events that can arrive in *any* time interval of length Δ, i.e., $\alpha^l(t-s) \leq R(t) - R(s) \leq \alpha^u(t-s)$ for all $t > s \geq 0$. In addition, $\alpha^l(0) = \alpha^u(0) = 0$. We also denote the tuple (α^l, α^u) with α. Service curves characterize the variability in the service provided by a resource. The curves $\beta^l(\Delta)$ and $\beta^u(\Delta)$ denote the minimum and the maximum number of events that can be processed within *any* time interval of length Δ, i.e., $\beta^l(t-s) \leq C(t) - C(s) \leq \beta^u(t-s)$ for all $t > s \geq 0$. In addition, $\beta^l(0) = \beta^u(0) = 0$. We also denote the tuple (β^l, β^u) with β. An event stream modeled by $\alpha(\Delta)$ enters a component and is processed using the resource modeled as $\beta^l(\Delta)$. The output is again an event stream $\alpha'(\Delta)$, and the *remaining* resource is expressed as $\beta'^l(\Delta)$. Note that the domain of the arrival and service curves are events, i.e., they describe the number of arriving events and the capability to process a certain number of events, respectively. The generalization towards physical quantities such as processing cycles or communication bits can be done by means of *workload curves* which is another instance of a VCC (see [6]).

2 Timing Analysis of Component Networks

In this section, we describe a timing analysis framework (in particular, for computing worst-case traversal times) for component networks that is based on Real-Time Calculus [3,7]. This calculus is an adaptation of Network Calculus [5] that was designed to analyze communication networks. Here, we consider three basic types of abstract components: Processing Element (PE), Playout Buffer (PB), and Earliest Deadline First (EDF) component. For the component models of Greedy Shapers, Time Division Multiple Access, servers, and hierarchical scheduling, please refer to [9,10,11]. In Sect.3 we lift this framework to an interface-theoretic setting and present our main technical result.

2.1 Processing Element

The PE component can be used to model a single processing element which processes one input stream. However, it can also be composed with other components of the same type, and model components processing more than one input stream using a fixed priority (FP) scheduling. Consider a concrete GP component that is triggered by the events of an incoming event stream. A fully preemptive task is instantiated at every event arrival to process the incoming event, and active tasks are processed in a FIFO order, while being restricted by the availability of resources. The completion of each task execution results in the corresponding event being removed from the input buffer and an event being emitted on the outgoing event stream.

Following results from Real-Time and Network Calculus [3,5], such a component can be modeled by an abstract component PE with the following internal component relations[1]:

$$\alpha'^u = \alpha^u \oslash \beta^l, \qquad \alpha'^l = \alpha^l \otimes \beta^l, \tag{1}$$

$$\beta'^l(\Delta) = \sup_{0 \le \lambda \le \Delta} \left\{ \beta^l(\lambda) - \alpha^u(\lambda) \right\} := RT(\beta^l, \alpha^u) \ , \tag{2}$$

and the backlog of the input buffer is bounded by $\sup_{0 \le \lambda \le \Delta} \left\{ \alpha^u(\lambda) - \beta^l(\lambda) \right\}$. If the available buffer space in the input buffer is constrained by $b_{\max}$, the backlog should never become bigger than the buffer size, i.e., we have a *buffer overflow constraint*. In this case, we can obtain the following component-based constraint on the admissible arrival and service curves:

$$\alpha^u(\Delta) \le \beta^l(\Delta) + b_{\max}, \quad \forall \Delta \in \mathbb{R}^{\ge 0} \ . \tag{3}$$

If the input arrival and service curves satisfy the above constraint, the backlog will never be bigger than $b_{\max}$. Before continuing with the computation of the WCTT for a PE component, we need to define the following shift function:

$$r(\alpha, c, \Delta) = \begin{cases} \alpha(\Delta - c) & \text{if } (\Delta > c) \wedge (\Delta \neq 0) \\ 0 & \text{if } (\Delta \le c) \vee (\Delta = 0) \end{cases} \tag{4}$$

which simply shifts a given curve $\alpha(\Delta)$ by the amount c to 'the right'. The WCTT experienced by an event in the component, defined as its finishing time minus its arrival time, can be computed as:

$$\mathrm{Del}(\alpha^u, \beta^l) := \sup_{\lambda \ge 0} \left\{ \inf\{\tau \ge 0 : \ \alpha^u(\lambda) \le \beta^l(\lambda + \tau)\} \right\} \ .$$

It is also possible to have systems with processing elements that process more than one data stream. For this purpose, the remaining service output of a higher priority PE component, computed with (2), can be connected to the service input of a lower priority PE component. This way we can model an FP resource sharing between PE components.

2.2 Playout Buffer

The PB component models a playout buffer. It receives data and stores it in a buffer which is read at a constant (usually periodic) rate. The buffer has a maximum size $B_{\max}$. We make the assumption that at the start of the system, there are already B_0 initial data items in the playout buffer, e.g., due to a playback delay. Data items in the playout buffer are removed at a constant rate. In particular, $P(t)$ data items are removed within the time interval $[0, t)$. This behavior can be described by the readout VCC $\rho(\Delta) = (\rho^l(\Delta), \rho^u(\Delta))$, i.e., $\rho^l(t - s) \le P(t) - P(s) \le \rho^u(t - s)$ for all $t > s \ge 0$. What needs to be

[1] See the Appendix for definitions of $\otimes$ and $\oslash$.

guaranteed is that the playout buffer never *overflows or underflows*. Following results on Real-Time and Rate Interfaces [4], for a PB component with input and readout event streams characterized by the VCCs α and ρ, respectively, and B_0 initial events, the playout buffer size $B(t)$ is constrained by $0 \leq B(t) \leq B_{\max}$ at all times if

$$\alpha^l(\Delta) \geq \rho^u(\Delta) - B_0, \text{ and } \alpha^u(\Delta) \leq \rho^l(\Delta) + B_{max} - B_0, \quad \forall \Delta \in \mathbb{R}^{\geq 0} \ . \quad (5)$$

The WCTT experienced by an event in the component can be computed as $\mathrm{Del}(\alpha^u, \rho^l_\tau)$ where

$$\rho^l_\tau(\Delta) = r(\rho^l, \tau, \Delta) \quad (6)$$

is the lower readout curve 'shifted to the right' by the initial playback delay $\tau \geq 0$ necessary to accumulate B_0 events.

2.3 Earliest Deadline First Component

The EDF component is similar to PE but it models processing of several data streams with a resource shared using the earliest deadline first scheduling policy. This requires a new abstract component with different internal relations [10]. Such a component processes N input event streams and emits N output event streams. Each input event stream i, $1 \leq i \leq N$, is associated with a fully preemptive task which is activated repeatedly by incoming events. Each input event stream i has an associated FIFO buffer with maximum size $b_{i\,max}$ where events are backlogged. Tasks process the head events in these buffers and are scheduled in an EDF order. Each task has a best-case execution time of BCET_i, a worst-case execution time WCET_i, and a relative deadline D_i where $0 \leq \mathrm{BCET}_i \leq \mathrm{WCET}_i \leq D_i$. The completion of a task execution results in the corresponding input event being removed from the associated buffer and an output event being emitted on the associated output event stream.

For an EDF component with a service curve β and event streams characterized by arrival curves α_i, all tasks are processed within their deadlines if and only if

$$\sum_{i=1}^{N} r(\alpha_i^u, D_i, \Delta) \leq \beta^l(\Delta), \quad \forall \Delta \in \mathbb{R}^{\geq 0} \ , \quad (7)$$

using the shift function r from (4). The output streams can be characterized by arrival curves computed for all streams i as:

$$\alpha_i'^u(\Delta) = r(\alpha_i^u, -(D_i - \mathrm{BCET}_i), \Delta), \quad \alpha_i'^l(\Delta) = r(\alpha_i^l, (D_i - \mathrm{BCET}_i), \Delta) \ , \quad (8)$$

and the number of events in input buffers do not exceed their capacity $b_{i\,\max}$ if

$$\alpha_i^u(D_i) \leq b_{i\,\max}, \quad \forall i \ . \quad (9)$$

The EDF component schedulability condition (7) can be related to the demand bound functions described in [2]. Given that the condition is satisfied, the service

curve provided to each stream can be modeled with a burst-delay function [5] defined for each stream i as:

$$\beta^l_{D_i}(\Delta) = \begin{cases} +\infty & \text{if } \Delta > D_i \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

The WCTT experienced by an event from a stream can be computed as $\mathrm{Del}(\alpha^u_i, \beta^l_{D_i})$ which is bounded by D_i for each stream.

2.4 Worst-Case Traversal Times of Component Networks

The worst-case traversal time of an event from an input stream which is processed by a sequence of components can be computed as the sum of the worst-case traversal times of the individual components. However, this would lead to a very pessimistic and unrealistic result as it would assume that the worst-case traversal times occur in all components for the same event. A better bound on the worst-case traversal time can be achieved by considering a concatenation of the components. This is a phenomenon known as "pay bursts only once" [5]. Following results from Network Calculus, this leads to the following computation for the WCTT.

For an input event stream α traversing a sequence of components which consists of a set of PEs, a set of PBs, and a set of EDF components denoted as $\mathcal{PE}$, $\mathcal{PB}$ and $\mathcal{EDF}$, respectively, the worst-case traversal time that an event can experience can be computed as $\mathrm{Del}(\alpha^u, \beta_{\mathcal{PE}} \otimes \rho_{\mathcal{PB}} \otimes \beta_{\mathcal{EDF}})$ with $\beta_{\mathcal{PE}} = \bigotimes_{c \in \mathcal{PE}} \beta^l_c$, $\rho_{\mathcal{PB}} = \bigotimes_{c \in \mathcal{PB}} \rho^l_{\tau\, c}$, $\beta_{\mathcal{EDF}} = \bigotimes_{c \in \mathcal{EDF}} \beta^l_{D_i\, c}$, and β^l_c is the service availability of PE component c, $\rho^l_{\tau\, c}$ is the lower readout curve for PB component c as defined with (6), and $\beta^l_{D_i\, c}$ is the service availability provided to the stream served with relative deadline D_i by EDF component c as defined with (10). A WCTT constraint on the sequence of components $\mathrm{Del}(\alpha^u, \beta_{\mathcal{PE}} \otimes \rho_{\mathcal{PB}} \otimes \beta_{\mathcal{EDF}} \otimes \sigma_{\mathcal{GS}}) \leq D$ can be written as follows:

$$\beta_{\mathcal{PE}} \otimes \rho_{\mathcal{PB}} \otimes \beta_{\mathcal{EDF}} \geq r(\alpha^u, D, \Delta), \quad \forall \Delta \in \mathbb{R}^{\geq 0} \ , \tag{11}$$

using the shift function r from (4).

3 Interface Algebra

In this section, we develop an interface-based design approach which will allow us by only inspecting the interfaces of two components to check whether WCTT and buffer underflow/overflow constraints would be satisfied if the components are composed together. The proposed interface algebra includes concepts from Real-Time Calculus, Assume/Guarantee Interfaces [1], and constraints propagation.

In our setup each component has two disjoint sets of input and output ports I and O. The actual input and output values of an abstract component are VCC curves. A connection from output j of one component to the input i of some other component will be denoted by (j, i). The interface of a component

makes certain *assumptions* on I, which are specified using the predicate $\phi^I(I)$. Provided this predicate is satisfied, the interface *guarantees* that the component works correctly and its outputs will satisfy a predicate $\phi^O(O)$.

In order to simplify the presentation, we introduce the *complies to* relation $\vdash$ between two VCC curves $a(\Delta)$ and $b(\Delta)$ as follows:

$$a \vdash b = (\forall \Delta : (a^l(\Delta) \geq b^l(\Delta)) \wedge (a^u(\Delta) \leq b^u(\Delta))) \ .$$

In other words, a complies to b ($a \vdash b$) if for all values of Δ the interval $[a^l(\Delta), a^u(\Delta)]$ is enclosed by $[b^l(\Delta), b^u(\Delta)]$.

Following the introduced notation, for any VCC α, we can define the input and output predicates for some component input i and output j as $\phi_i^I(\alpha_i) = (\alpha_i \vdash \alpha_i^A)$ and $\phi_j^O(\alpha_j) = (\alpha_j \vdash \alpha_j^G)$, respectively, where α^A and α^G are assume and guarantee curves provided by the component interface.

We would like to have that if the input predicates of a component are all satisfied, then it works correctly and all output predicates are satisfied. In other words the condition $\bigwedge_{\forall i \in I} \phi_i^I(\alpha_i) \Rightarrow \bigwedge_{\forall j \in J} \phi_j^O(\alpha_j)$ must be satisfied by the interfaces of all components.

If we now connect several components, we want to be able to check if the whole system can work correctly by just checking whether their interfaces are compatible. This can be done by testing whether the relation $\bigwedge_{\forall (j,i)} \phi_j^O(\alpha_j) \Rightarrow \bigwedge_{\forall (j,i)} \phi_i^I(\alpha_i)$ is satisfiable. In other words, we must check if there exists *some* environment in which the components can be composed. The relation is hence the weakest precondition on the environment of the system.

We also need to propagate information about the predicates between the interfaces, see also [8]. This way, we combine interface theory with constraints propagation, which enables parameterized design of component-based systems. We propagate the assume and guarantee curves of the input and output predicates through the interfaces. Each interface connection would have both assume and guarantee curves propagated in opposite directions. We connect the interfaces, i.e., the corresponding guarantee and assume curves, as $\forall (j,i) : \ (\alpha^G(i) = \alpha^G(j)) \wedge (\alpha^A(j) = \alpha^A(i))$.

Now, we can determine whether two abstract components are compatible by checking the compatibility of their interfaces. Let us suppose that the assume and guarantee variables of an interface of any component and their relation to the input and output values of the corresponding abstract component satisfy

$$(\forall i \in I : \ \alpha_i \vdash \alpha_i^G \vdash \alpha_i^A) \Rightarrow (\forall j \in J : \ \alpha_j \vdash \alpha_j^G \vdash \alpha_j^A) \ , \tag{12}$$

where the component has inputs I and outputs J. Then if for a network of components, the relation $\alpha_i^G \vdash \alpha_i^A$ is satisfied for all inputs i, we can conclude that the system works correctly.

Now we need to develop the relations between guarantees and assumptions in order to satisfy (12) for every component. We will first describe a general method how these relations can be determined and then apply it to the abstract components described so far.

To this end, as we are dealing with stateless interfaces, I and O can be related by a *transfer function*, e.g., $O = F(I)$. The actual function depends on the processing semantics of the modeled component.

We need to define the concept of a monotone abstract component. Note that the 'complies to' relation $\vdash$ has been generalized to tuples, i.e., $(a_i : i \in I) \vdash (b_i : i \in I)$ equals $\forall i \in I : \ a_i \vdash b_i$.

Definition 1. *An abstract component with a set of input and output ports, I and J, respectively, and a transfer function F that maps input curves to output curves, is* monotone *if* $((\tilde{\alpha}_i : i \in I) \vdash (\alpha_i : i \in I)) \Rightarrow ((\tilde{\alpha}_j : j \in J) \vdash (\alpha_j : j \in J))$ *where* $(\alpha_j : j \in J) = F(\alpha_i : i \in I)$ *and* $(\tilde{\alpha}_j : j \in J) = F(\tilde{\alpha}_i : i \in I)$.

In other words, if we replace the input curves of an abstract component with curves that are compliant, then the new output curves are also compliant to the previous ones. Note that all components we look at in this paper satisfy this monotonicity condition, see for example (1), (2), and (8).

The following theorem leads to a constructive way to compute the input assumes and output guarantees from the given input guarantees and output assumes. We make use of the individual components of the transfer function F, i.e., $\alpha_j = F_j(\alpha_i : i \in I)$ for all $j \in J$ where I and J denote the input and output ports of the corresponding abstract component, respectively. The theorem establishes that we can simply determine the output guarantees using the components of a given transfer function of an abstract component. For the input assumes we need to determine inverses of the transfer function F_j with respect to at least one of its arguments. All arguments of some F_j are determined by the input guarantees but one, say for example $\alpha^G_{i^*}$. This one we replace by $\alpha^A_{i^*}$ and try to determine this curve such that the result of the transfer function still complies to the given output assumes. If we choose the same i^* for several components of the output function, then the resulting $\alpha^A_{i^*}$ needs to comply to all partial 'inverses'.

Theorem 1. *Given a monotone component with input ports I, output ports J, and a transfer function F that maps input curves to output curves, i.e., $(\alpha_j : j \in J) = F(\alpha_i : i \in I)$. Let us suppose that we determine the output guarantees using:*

$$\alpha^G_j = F_j(\alpha^G_i : i \in I) \quad \forall j \in J \ , \tag{13}$$

and the input assumes are computed such that

$$\forall j \in J \ \ \exists i^* \in I : \left(F_j(\alpha^G_i : i \in I)\Big|_{\alpha^G_{i^*} \leftarrow \alpha^A_{i^*}} \ \vdash \ \alpha^A_j \right) \ , \tag{14}$$

where $\alpha^G_{i^*} \leftarrow \alpha^A_{i^*}$ denotes that in the preceding term $\alpha^G_{i^*}$ is replaced by $\alpha^A_{i^*}$.

Then (12) holds.

Proof. Let us assume that for all input ports $i \in I$ we have $\alpha_i \vdash \alpha^G_i$, see (12). Using the monotonicity of F, we can now see that $(\forall i \in I : \alpha_i \vdash \alpha^G_i) \Rightarrow F(\alpha_i : i \in I) \vdash F(\alpha^G_i : i \in I) \Rightarrow (\forall j \in J : \alpha_j \vdash \alpha^G_j)$.

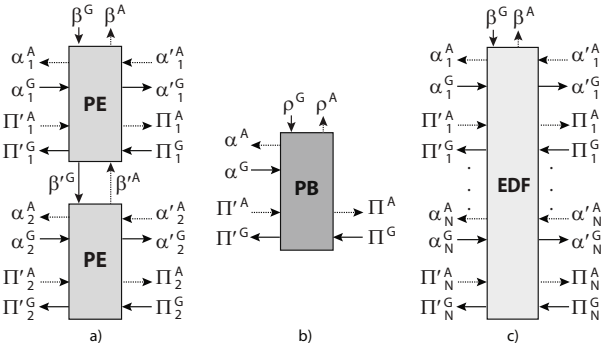


Fig. 1. Interface models for: a) two PE components processing two streams with FP scheduling b) PB component c) EDF component processing N streams

We still need to show that $(\forall i \in I : \alpha_i^G \vdash \alpha_i^A) \Rightarrow (\forall j \in J : \alpha_j^G \vdash \alpha_j^A)$ using the construction in (13). At first note that this expression is equivalent to $\forall j \in J \;\; \exists i^* \in I : \big((\alpha_{i^*}^G \vdash \alpha_{i^*}^A) \Rightarrow (\alpha_j^G \vdash \alpha_j^A)\big)$. We also know that for any $i^* \in I$ we have $(\alpha_{i^*}^G \vdash \alpha_{i^*}^A) \Rightarrow ((\alpha_i^G : i \in I) \vdash (\alpha_i^G : i \in I) \mid_{\alpha_{i^*}^G \leftarrow \alpha_{i^*}^A})$.

Because of the monotonicity of F we can derive that for any $i^* \in I$ we have $(\alpha_{i^*}^G \vdash \alpha_{i^*}^A) \Rightarrow (F(\alpha_i^G : i \in I) \vdash F(\alpha_i^G : i \in I) \mid_{\alpha_{i^*}^G \leftarrow \alpha_{i^*}^A})$, and using (13) we find $\forall j \in J \;\; \exists i^* \in I$ such that $((\alpha_{i^*}^G \vdash \alpha_{i^*}^A) \Rightarrow (F_j(\alpha_i^G : i \in I) \vdash F_j(\alpha_i^G : i \in I) \mid_{\alpha_{i^*}^G \leftarrow \alpha_{i^*}^A}) \Rightarrow (\alpha_j^G \vdash \alpha_j^A))$. □

Next, we show how to compute the largest upper curve and smallest lower curve for which the respective relations still hold. This leads to the weakest possible input assumptions. We do this for the three types of components introduced so far.

3.1 Processing Element

Now, using the relation between interface values, assumptions and guarantees in (12), and following the results from Theorem 1, we can deduce that the equations describing the output guarantees are equivalent to those for the abstract component, i.e., (1), just using interface guarantees instead of values. Therefore, we have:

$$\alpha'^{uG} = \alpha^{uG} \oslash \beta^{lG}, \qquad \alpha'^{lG} = \alpha^{lG} \otimes \beta^{lG} \;\; .$$

In order to calculate the input assumptions of the PE abstract component, we need to determine inverse relations corresponding to (1) and (3). Following results from Network Calculus [5], we can do this by determining the pseudo-inverse functions which have the following definition $f^{-1}(x) = \inf\{t : f(t) \geq x\}$.

In order to guarantee that all relations hold if the input and output predicates are satisfied, we then need to use the minimum (in case of the upper curves) or the maximum (in case of the lower curves) of all the determined pseudo-inverses.

From the pseudo-inverses of (1), we get the inequalities $\alpha^{lA} \geq \alpha'^{lA} \oslash \beta^{lG}$ and $\beta^{lA} \geq \alpha'^{lA} \oslash \alpha^{lG}$. Here we use the duality relation between the $\oslash$ and $\otimes$ operators (see the Appendix). Similarly, we get the inequalities $\beta^{lA} \geq \alpha^{uG} \oslash \alpha'^{uA}$ and $\alpha^{uA} \leq \beta^{lG} \otimes \alpha'^{uA}$. Inverting the buffer overflow constraint (3) is trivial and we get the inequalities $\alpha^{uA} \leq \beta^{lG} + b$ and $\beta^{lA} \geq \alpha^{uG} - b_{max}$.

If a PE component shares the service it receives with other lower priority PE components, the remaining service is bounded by (2). In terms of output guaranteed values, this can be expressed as $\beta'^{lG}(\Delta) = RT(\beta^{lG}, \alpha^{uG})$ where the RT operator is defined in (2). In order to obtain the input assumptions of a component using FP scheduling, we need to use the inverses of the RT operator (see the Appendix).

After combining all inverses, the assumptions related to component PE can be determined as follows:

$$\alpha^{uA} = \min\{\beta^{lG} \otimes \alpha'^{uA}, \beta^{lG} + b_{\max}, RT^{-\alpha}(\beta'^{lA}, \beta^{lG})\}, \quad \alpha^{lA} = \alpha'^{lA} \oslash \beta^{lG},$$
$$\beta^{lA} = \max\{\alpha'^{lA} \oslash \alpha^{lG}, \alpha^{uG} \oslash \alpha'^{uA}, \alpha^{uG} - b_{\max}, RT^{-\beta}(\beta'^{lA}, \alpha^{uG})\} \ . \qquad (15)$$

The interface connections for two PE components are illustrated in Fig. 1a.

3.2 Playout Buffer

For a PB component, the relations are simpler. We only need to determine the inverse relations for the buffer constraints (5), which directly yield the following relations:

$$\alpha^{uA} = \rho^{lG} + B_{max} - B_0, \qquad \alpha^{lA} = \rho^{uG} - B_0,$$
$$\rho^{uA} = \alpha^{lG} + B_0, \qquad \rho^{lA} = \alpha^{uG} - (B_{max} - B_0) \ . \qquad (16)$$

The interface connections for a single PB component are illustrated in Fig. 1b.

3.3 Earliest Deadline First Component

Similarly to the PE component, equations describing the output guarantees are again equivalent to those for the abstract component, i.e., (8). They only need to be expressed in terms of interface variables instead of values for all streams i:

$$\alpha_i'^{uG}(\Delta) = r(\alpha_i^{uG}, -(D_i - \mathrm{BCET}_i), \Delta), \quad \alpha_i'^{lG}(\Delta) = r(\alpha_i^{lG}, (D_i - \mathrm{BCET}_i), \Delta) \ ,$$

using the definition of the shift function r in (4).

Similarly, for the resource and buffer constraints, (7) and (9), we obtain:

$$\sum_{i=1}^{N} r(\alpha_i^{uG}, D_i, \Delta) \leq \beta^{lG}(\Delta), \quad \forall \Delta \in \mathbb{R}^{\geq 0},$$
$$\alpha_i^{uG}(D_i) \leq b_{i\,max}, \quad \forall i \ .$$

Determining the input assumptions of the EDF component also involves finding the pseudo-inverse functions of the relations. Finding the input assumes for the upper arrival curves involves inverting (7) and (9). Again, we need to compute the largest upper curves for which the relations still hold. Finding the inverses and combing them, we find for all streams i:

$$\alpha_i^{uA}(\Delta) = \min\Big\{\beta^{lG}(\Delta + D_i) - \sum_{\substack{j=1 \\ j\neq i}}^{N} r(\alpha_j^{uG}, (D_j - D_i), \Delta), \\ s(\alpha_i'^{uA}, (D_i - \mathrm{BCET}_i), \Delta),\ t(D_i, b_{i\,max}, \Delta)\Big\}\ ,$$

using functions $s(\alpha, c, \Delta)$ and $t(d, b, \Delta)$ defined as:

$$s(\alpha, c, \Delta) = \begin{cases} \alpha(\Delta - c) & \text{if } \Delta > c \\ \lim_{\epsilon\to 0}\{\alpha(\epsilon)\} & \text{if } 0 < \Delta \leq c \\ 0 & \text{if } \Delta = 0 \end{cases} \qquad t(d, b, \Delta) = \begin{cases} \infty & \text{if } \Delta > d \\ b & \text{if } 0 < \Delta \leq d \\ 0 & \text{if } \Delta = 0 \end{cases}$$

Calculating the input assumption for the lower curve is much simpler as it involves finding the smallest lower curve solution to the pseudo-inverse of (8) or $\alpha_i^{lA}(\Delta) \geq \alpha_i'^{lA}(\Delta + (D_i - \mathrm{BCET}_i))$ for all i. Therefore, we can determine the following assume interface function for the lower curve of each input data stream:

$$\alpha_i^{lA}(\Delta) = r(\alpha_i'^{lA}, -(D_i - \mathrm{BCET}_i), \Delta), \quad \forall i\ ,$$

using the shift function r as defined in (4).

Similarly, for the assume of the lower service curve we invert (7) which yields the inequality $\beta^{lA}(\Delta) \geq \sum_{i=1}^{N} r(\alpha_i^{uG}, D_i, \Delta)$. Therefore, the input assume for the lower service curve of an EDF component can be determined as:

$$\beta^{lA}(\Delta) = \sum_{i=1}^{N} r(\alpha_i^{uG}, D_i, \Delta)\ . \tag{17}$$

The interface model for the EDF component is illustrated in Fig 1c.

3.4 Worst-Case Traversal Time Interface

We develop an additional type of interface to alleviate design of systems with WCTT constraints. It is an interface-based interpretation of the analytical computation of WCTT with (11).

The 'complies to' relation $\vdash$ for this interface connection is defined as $\Pi^G(\Delta) \vdash \Pi^A(\Delta) = (\forall \Delta : \Pi^G(\Delta) \geq \Pi^A(\Delta))$, where Π^A expresses the minimum service requested from all subsequent components such that the WCTT constraint is satisfied, and Π^G expresses the minimum service guaranteed by all subsequent components.

Computing the guarantee for a sequence of components follows directly from (11) and can be done with $\Pi^G = \beta^G_{\mathcal{PE}} \otimes \rho^G_{\mathcal{PB}} \otimes \beta^G_{\mathcal{ED}\mathcal{F}}$. Connecting a PE component

to the sequence would change the combined service to $\Pi'^G = \beta^{lG} \otimes \Pi^G$ where β^{lG} is the lower service guaranteed by the PE. Similarly, connecting a PB component we would have $\Pi'^G = \rho_\tau^{lG} \otimes \Pi^G$, where $\rho_\tau^l(\Delta)$ is the lower guaranteed shifted readout curve as defined with (6). For an EDF component, we have $\Pi'^G = \beta_{D_i}^{lG} \otimes \Pi^G$ where $\beta_{D_i}^{lG}$ is the service curve for the stream with relative deadline D_i as defined in (10).

Inverting (11), we can compute the assume on the combined service of a sequence of components as $\Pi^A = r(\alpha^{uG}, D, \Delta)$ which expresses the minimum necessary service in order to meet a WCTT constraint of D for the input α^{uG}. Propagating the assume value through a sequence of components can be done for the three types of components by inverting (11) as follows:

$$\text{PE: } \Pi^A = \Pi'^A \oslash \beta^{lG}, \quad \text{PB: } \Pi^A = \Pi'^A \oslash \rho_\tau^{lG}, \quad \text{EDF: } \Pi^A = \Pi'^A \oslash \beta_{D_i}^{lG} \ .$$

We can also compute component-wise constraints on the resources provided by each component given the resource assumption from preceding components Π'^A, and the resource guarantee from subsequent components Π^G:

$$\text{PE: } \beta^{lA} \geq \Pi'^A \oslash \Pi^G, \quad \text{PB: } \rho_\tau^{lA} \geq \Pi'^A \oslash \Pi^G, \quad \text{EDF: } \beta_{D_i}^{lA} \geq \Pi'^A \oslash \Pi^G \ .$$

The above constraints can be combined with the previously computed input assumes for the resources of the three components with (15), (16), and (17). By doing this, satisfying all interface relations of components composed in a sequence will guarantee that the WCTT constraint on the sequence of components is satisfied. The WCTT interfaces for the PE, PB, and EDF components are shown in Fig.1.

4 Illustrative Example

In this section we show how our proposed theory can be applied to an example system shown in Fig.2. Towards this, each PE, PB, and EDF component is considered to be an independent component, and our objective is to connect them together to realize the architecture shown in the figure. In order to decide whether two components can be connected together, we would only inspect their interfaces. Two compatible interfaces implicitly guarantee that the buffers inside their respective components will never overflow or underflow, and in addition, the WCTT constraints are satisfied.

The main message in this section is an illustration of how the internal details of a component (e.g., its buffer size, scheduling policy, processor frequency, deadline) are reflected (or summarized) through its interfaces. We show that if these internal details are changed then the component's interfaces also change and two previously compatible components may become incompatible (or vice versa).

Experimental Setup. We consider the system illustrated in Fig.2. It consists of a multiprocessor platform with four CPUs. A distributed application is mapped to the platform. It processes two data streams, a high priority one denoted as

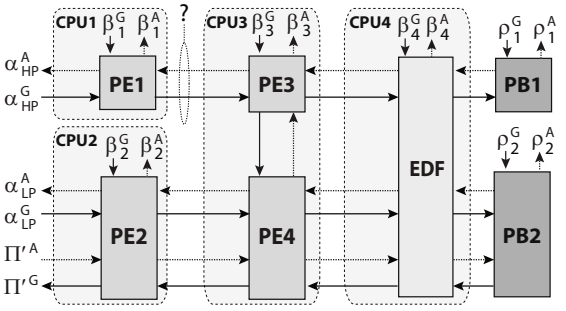


Fig. 2. Interface model of an example stream processing system

HP, and a low priority one denoted as LP. The application consists of six tasks. Streams are preprocessed by the tasks modeled with components $PE1$ and $PE2$ which are mapped separately to $CPU1$ and $CPU2$, respectively. Afterwards, they are processed by components $PE3$ and $PE4$ which are mapped to $CPU3$. The tasks share the resource using FP scheduling where stream HP is given higher priority. Additionally, streams are processed by two tasks mapped to $CPU4$ which they share with the EDF policy. This is modeled with the EDF component. The fully processed streams are written to playout buffers which are read by an I/O interface at a constant rate. The buffers are modeled with components $PB1$ and $PB2$. For simplicity, the communication is not modeled here. If necessary, it can be taken into account by additional components in the model.

Data packets from the streams have bursty arrivals described with period p, jitter j, and minimum inter-arrival distance d. For the HP stream the parameters are $p = 25$ms, $j = 40$ms, $d = 0.1$ms, and for LP stream they are $p = 25$ms, $j = 30$ms, $d = 0.1$ms. Each data packet from the two streams has a constant processing demand of 1 cycle. $CPU1$ is fully available with service availability of 0.3 cycles/ms. For $CPU2$, $CPU3$, and $CPU4$, the respective service availabilities are 0.3, 0.4, and 0.4 cycles/ms. Components $PE3$ and $PE4$ have internal buffer sizes of 2 and 3 packets, respectively. The buffers should never overflow. The EDF component schedules tasks processing streams HP and LP with relative deadlines of 8ms and 10ms, respectively, with both buffers being limited to 3 packets. These buffers should also never overflow. Components $PB1$ and $PB2$ are read at a constant rate of 25 packets/ms. Both components have maximum buffer sizes of 8 data packets, and initially they contain 4 data packets. Both buffers should not underflow and overflow. Additionally, we have a WCTT constraint on the LP stream of 200ms.

Results. We consider three different scenarios of the system's parameters. In each of them, we check the compatibility of component $PE1$ with the partially designed system when all other components are already connected. Compatibility is checked by only inspecting the interface connection between $PE1$ and the

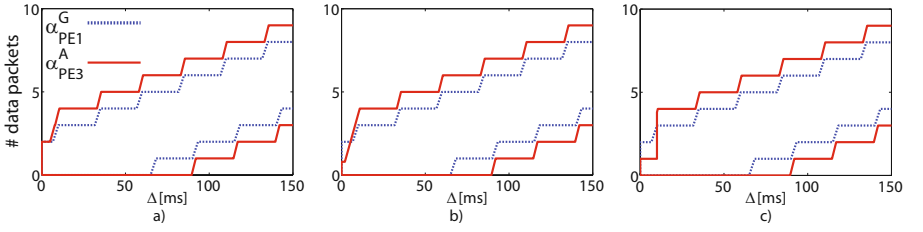


Fig. 3. Interface connection between the output guarantee of component $PE1$ and the input assumption of component $PE3$ shows: a) compatibility b) incompatibility when WCTT for stream LP is reduced to 192ms c) incompatibility when buffer of component $PB2$ is decreased to 5 packets.

system which is marked with '?' in Fig.2. Compatibility meaning that the output guarantee is fully "enclosed" by the input assumption.

Case I: The system is considered with the specified parameters. The components turn out to be compatible. The interface connection is illustrated in Fig.3a. It shows that the guarantee on the output stream rate α^G_{PE1} expressed by $PE1$s interface is compatible with the assumption on the input rate α^A_{PE3} expressed by $PE3$s interface.

Case II: The WCTT constraint on the LP stream is decreased to 192ms. This leads to incompatibility between components $PE1$ and $PE3$ which reveals in the interface connection as shown in Fig.3b.

Case III: The maximum buffer size of component $PB2$ is decreased to 5 packets which leads to incompatibility as shown in Fig.3c.

In summary, we have shown through a concrete example how incremental compatibility checking can be done using the proposed interfaces. Clearly, such interfaces can also be used in a straightforward manner for resource dimensioning and component-level design space exploration. Typical questions that one would ask are: What is the minimum buffer size of a component such that its interface is compatible with a partially existing design? What is the minimum processing frequency such that the interface is still compatible? Or what are the feasible relative deadlines in an EDF component? In this paper, we are concerned with buffer and WCTT constraints however, one can imagine developing similar interfaces for power, energy, and temperature constraints.

5 Concluding Remarks

In this paper we proposed an interface algebra for checking whether multiple components of an embedded system may be composed together while satisfying their worst-case traversal time (WCTT) constraints. The main advantage of such an interface-based formulation is that component composition only requires a compatibility checking of the interfaces of the components involved, without

having to compute the WCTT of the entire component network from scratch, each time a new component is added or an existing component is modified. This has a number of advantages. It significantly reduces design complexity, it does not require components to expose the details of their internals, and it allows a correct-by-construction design flow.

The interfaces studied here were purely functional in nature, i.e., they do not contain any state information. This might be restrictive in a number of settings, e.g., when the components implement complex protocols. As an example, the processing rate of a component might depend on the "state" or the fill level of an internal buffer. As a part of future work, we plan to extend our interface algebra to accommodate such "stateful" components. This may be done by describing an automaton to represent an interface, with language inclusion or equivalence to denote the notion of *compatibility* between components.

References

1. de Alfaro, L., Henzinger, T.A.: Interface theories for component-based design. In: Henzinger, T.A., Kirsch, C.M. (eds.) EMSOFT 2001. LNCS, vol. 2211, pp. 148–165. Springer, Heidelberg (2001)
2. Baruah, S., Chen, D., Gorinsky, S., Mok, A.: Generalized multiframe tasks. Real-Time Systems 17(1), 5–22 (1999)
3. Chakraborty, S., Kunzli, S., Thiele, L.: A general framework for analysing system properties in platform-based embedded system designs. In: Design, Automation and Test in Europe (DATE), p. 10190 (2003)
4. Chakraborty, S., Liu, Y., Stoimenov, N., Thiele, L., Wandeler, E.: Interface-based rate analysis of embedded systems. In: 27th IEEE International Real-Time Systems Symposium (RTSS), pp. 25–34 (2006)
5. Le Boudec, J.Y., Thiran, P.: Network Calculus: A Theory of Deterministic Queuing Systems for the Internet. Springer, Heidelberg (2001)
6. Maxiaguine, A., Künzli, S., Thiele, L.: Workload characterization model for tasks with variable execution demand. In: Design, Automation and Test in Europe (DATE), p. 21040 (2004)
7. Thiele, L., Chakraborty, S., Naedele, M.: Real-time calculus for scheduling hard real-time systems. In: IEEE International Symposium on Circuits and Systems (ISCAS), vol. 4, pp. 101–104 (2000)
8. Thiele, L., Wandeler, E., Stoimenov, N.: Real-time interfaces for composing real-time systems. In: 6th ACM & IEEE International Conference on Embedded Software (EMSOFT), pp. 34–43 (2006)
9. Wandeler, E., Maxiaguine, A., Thiele, L.: Performance analysis of greedy shapers in real-time systems. In: Design, Automation and Test in Europe (DATE), pp. 444–449 (2006)
10. Wandeler, E., Thiele, L.: Interface-based design of real-time systems with hierarchical scheduling. In: 12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), pp. 243–252 (2006)
11. Wandeler, E., Thiele, L.: Optimal TDMA time slot and cycle length allocation for hard real-time systems. In: Asia and South Pacific Design Automation Conference (ASP-DAC), pp. 479–484 (2006)

Appendix: Min-Max Algebra

The min-plus algebra convolution and deconvolution operators are defined as:

$$(f \otimes g)(\Delta) = \inf_{0 \le \lambda \le \Delta} \{f(\Delta - \lambda) + g(\lambda)\}, \quad (f \oslash g)(\Delta) = \sup_{\lambda \ge 0} \{f(\Delta + \lambda) - g(\lambda)\}.$$

The duality between $\otimes$ and $\oslash$ states that: $f \oslash g \le h \iff f \le g \otimes h$.
The inverses of the $RT(\beta, \alpha)$ are defined as:

$$\alpha = RT^{-\alpha}(\beta', \beta) \Rightarrow \beta' \le RT(\beta, \alpha), \quad \beta = RT^{-\beta}(\beta', \alpha) \Rightarrow \beta' \le RT(\beta, \alpha),$$

with solutions:

$$RT^{-\alpha}(\beta', \beta)(\Delta) = \beta(\Delta + \lambda) - \beta'(\Delta + \lambda) \quad \text{for } \lambda = \sup\{\tau \ : \ \beta'(\Delta + \tau) = \beta'(\Delta)\},$$
$$RT^{-\beta}(\beta', \alpha)(\Delta) = \beta'(\Delta - \lambda) + \alpha(\Delta - \lambda) \quad \text{for } \lambda = \sup\{\tau \ : \ \beta'(\Delta - \tau) = \beta'(\Delta)\}.$$

Towards Resource-Optimal Routing Plans for Real-Time Traffic

Alessandro Lori, Giovanni Stea, and Gigliola Vaglini

Dipartimento di Ingegneria dellíInformazione, University of Pisa
Via Diotisalvi 2, I-56122 Pisa, Italy
{a.lori,g.stea,g.vaglini}@iet.unipi.it

Abstract. We discuss the issue of computing resource-optimal routing plans in a network domain. Given a number of known traffic demands, with associated required delays, we discuss how to route them and allocate resources for them at each node so that the demands are satisfied. While a *globally* optimal routing plan requires joint computation of the paths and of the associated resources (which was claimed to be NP-hard), in this paper we stick to existing approaches for path computation, and use mathematical programming to model resource allocation once the paths are computed. We show that the problem is either convex or non-convex, depending on the scheduling algorithms adopted at the nodes. Our results show that, by computing resources per-path, instead of globally, the available capacity can be exceeded even at surprisingly low utilizations.

1 Introduction

Real-time traffic over IP networks has become a reality. Several applications, e.g. industrial control, remote sensing and surveillance systems, live IPTV and VoIP etc., all requiring real-time guarantees (i.e., a bound on the end-to-end delay) are increasingly being deployed. Internet Service Providers are already facing, or will soon face, the challenge of configuring their network domains so as to provide deterministic delay bound guarantees to their customers – whether single users or lower-tier providers themselves – by negotiating real-time oriented Service Level Agreements (SLAs). Supporting SLAs with real-time constraints requires proper Traffic Engineering (TE) and resource optimization practices. Multi-Protocol Label Switching (MPLS, [2]) with TE extensions (MPLS-TE, [17]) allows traffic trunks to be routed along arbitrary paths, and resources to be allocated on those paths at the time of flow setup.

Supporting delay constrained traffic requires in fact *both* computing paths *and* reserving resources along those paths. The usual assumption (to which we stick in the rest of the paper) is that traffic trunks are scheduled at each node so as to be reserved a minimum guaranteed rate. As far as path computation is concerned, a relevant amount of literature has been published since the late ‘90s, under the name of QoS routing, a good review of which can be found in [25]. Most papers (see, e.g. [21-23]), assume that delays are *static* and/or *additive* per-link metrics. However, delay bounds do depend on the amount of reserved resources at each link, i.e. on the number and amount of flows traversing them, and the expression of the delay bound is not linear

T. Margaria and B. Steffen (Eds.): ISoLA 2010, Part I, LNCS 6415, pp. 214–227, 2010.

in the number of links. Other papers tackle the problem from a probabilistic point of view, assuming a stochastic characterization of traffic and attempting to minimize or bound the *average* delay, which is hardly relevant for real-time traffic (e.g. [25]). A limited number of works [18-19] propose path computation techniques constrained by deterministic (non additive) delay bound constraints, taking resource allocation into account. [18] shows that it is possible to compute a shortest path for a *single* flow, subject to end-to-end delay bounds, also computing the rate to be reserved on each node during path computation, at a polynomial cost. It also assumes that an equal rate has to be reserved at each node for the path. [19] proposes lower-complexity approximate solutions to the problem solved exactly in [18].

As far as resource allocation is concerned, the problem is often referred to as *QoS partitioning* in the literature. On that topic, several works exist that achieve optimal partitions for *additive* delays on a given path (see, e.g., [25]). An interesting work [20] shows that, when using end-to-end delay bounds as constraints, reserving the *same* rate (as done in [18-19]) may be suboptimal and lead to failing of paths which might indeed be admissible. Authors propose an algorithm that allows a delay-feasible resource allocation to be computed *on a given path*, if such an allocation exists.

To the best of our knowledge, the problem of making a *global* routing and resource allocation plan under delay bound constraints has received little attention so far. [19] claims that the problem is NP-hard[1]. However, this does not mean that it is not solvable for practical dimensions (i.e., comparable to those of today's and tomorrow's network domains), nor it implies that good suboptimal solutions cannot be computed in reasonable time, even for large dimensions. Besides, global routing plans do not need to be computed in real time. Network engineering and optimization cycles – where new routing plans are made from scratch, based on the traffic forecast and negotiated SLAs – do not take place more frequently than daily or weekly, hence computation time can be traded for optimality. Second, per-path computation and resource allocation is feasible in a dynamic environment (*online TE*), but is clearly suboptimal when routing plans are considered (*offline TE*).

In this paper we mark a first step in this direction by tackling *global* resource allocation with delay bound constraints in a network domain. We assume that paths have been selected (and we evaluate several existing options for the path computation phase), and we exploit optimization techniques to minimize the amount of rate reserved in the network domain. We show that the problem can be solved optimally for several classes of schedulers. For some schedulers it has a convex formulation, which means that optimal solutions can be found in a reasonable time. Our first results show that, even at surprisingly low network loads, global allocation is necessary to be able to guarantee delay bounds when it is indeed feasible to do so, as per-path solutions are generally ineffective.

While in this work we assume that routing is given, and we only aim at optimizing the resource allocation, the long-term goal of this stream of research is to provide effective algorithms for *joint* path computation and resource reservation in a network domain, which is actively being pursued at the time of writing.

The rest of the paper is organized as follows: in section 2 we describe the system model. We formulate the resource optimization problem in Section 3. Section 4

[1] The paper reports the statement as "proposition 1", without a proof or a reference to one.

reports numerical results related to a case-study network. We conclude the paper in Section 5.

2 System Model

We represent a network domain through a graph $G \equiv \{V, E\}$, where V is a set of *nodes*, i.e. routers, and $E \equiv \{(i,j) : i,j \in V, i \neq j\}$ are a set of directed *links*. We assume that links are bidirectional, so that $(i,j) \in E \Rightarrow (j,i) \in E$. Each link is characterized by a physical link speed $W_{(i,j)}$, a propagation delay $pd_{(i,j)}$, both constant, and a reservable TE capacity $C_{(i,j)} \leq W_{(i,j)}$. While it is normally $W_{(i,j)} = W_{(j,i)}$ and $pd_{(i,j)} = pd_{(j,i)}$ due to technological constraints, the same cannot be said a priori regarding TE capacities without this affecting the generality. Nodes may have a constant *transit delay* td_x, $x \in V$. This does not include queuing delays, which are variable and considered separately. Routers are further distinguished into *core* and *edge* nodes. Let $B \subseteq V$ be the set of *edge* routers, so that $F = \{B \times B\} \setminus \{(i,i), i \in B\}$ denotes the possible *routes* for traffic flowing through the domain. Note that we can account for local destinations inside the domain by including into B those LSRs where traffic is originated/destined. Furthermore, note that – in general – the same route can be connected by traversing different *paths,* i.e. sequences of nodes and links. Define a *path* that connects the route (i,e) as:

$$P_{(i,e)} = \left\{ \begin{array}{l} (x_j, y_j) \in E, 1 \leq j \leq N_{(i,e)} : \\ x_1 = i,\ y_{N_{(i,e)}} = e, \forall j : 1 \leq j \leq N_{(i,e)} - 1\ y_j = x_{j+1} \end{array} \right\}$$

i.e. a set of links that connect node i to node e. We are only interested in *loop-free* paths, i.e. those for which $j \neq k \Leftrightarrow y_j \neq y_k$. Note that the above formulation allows paths to be arbitrary, i.e. not to form a tree based at the destination node (as would happen instead with destination-based forwarding).

Within the domain, traffic trunks or *flows* have to be accommodated. The latter are characterized by a route $(i,e) \in F$, which denotes their ingress and egress points, a traffic constraint in the form of leaky-bucket parameters $\sigma_{(i,e)}, \rho_{(i,e)}$, and a *required delay bound* $\delta_{(i,e)}$. For the sake of readability (i.e., to avoid adding further subscripts), we describe the model under the assumption that *one* flow exists for a given route. The alert reader will easily notice that multiple flows on the same route can be accounted for in this model.

We assume that each link is managed by a *packet scheduler*, which arbitrates packets of different flows according to their *reserved rates*. The only assumption that we make on the scheduling algorithms is that they can be modeled via *rate-latency service curves* [3]. Several types of commonplace schedulers fit into this category, from

Packet Generalized Processor Sharing (PGPS, [4]) to Worst-case Fair Weighted Fair Queuing (WF^2Q, [15]), to Self Clocked Fair Queuing (SCFQ, [10]), to Deficit Round Robin (DRR, [5]). This kind of service curves, however, leaves out some popular schedulers, such as the well-known Earliest Deadline First (EDF), which have often been used in connection with QoS partitioning problems [24]. Schedulers need not be the same at each link for the model to be valid: nevertheless, we will often assume so when performing the analysis, especially to show that different schedulers lead to different performance. A rate-latency scheduler is able to divide the reservable TE capacity among the flows, giving to each one a reserved (long-term) rate, subject to a short-term vacation called *latency*. We denote with $R_{(x,y),(i,e)}$ the reserved rate at link (x,y) for flow (i,e). The latter may be null, for instance if flow (i,e) does not traverse link (x,y). TE capacity constraints need be accounted for, which is done by ensuring that:

$$\sum_{(i,e)\in F} R_{(x,y),(i,e)} \leq C_{(x,y)} \tag{1}$$

The latency is denoted by $\theta_{(x,y),(i,e)}$, and it is monotonically decreasing with the reserved rate. The exact expression for the latency is scheduler-specific. We will come back to this later on, showing that rate-latency schedulers may fall into three categories, thus giving birth to slightly different formulations for the problem.

Whatever the expression for the latency, the end-to-end delay for a flow along path $P_{(i,e)}$ is the following:

$$D_{P_{(i,e)}} = \sum_{(x,y)\in P_{(i,e)}} \left[\theta_{(x,y),(i,e)} + tp_{(x,y)} + td_x\right] + \frac{\sigma_{(i,e)}}{\min\limits_{(x,y)\in P_{(i,e)}} \left\{R_{(x,y),(i,e)}\right\}}, \tag{2}$$

provided that:

$$\min_{(x,y)\in P_{(i,e)}} \left\{R_{(x,y),(i,e)}\right\} \geq \rho_{(i,e)},$$

and $D_{P_{(i,e)}} = \infty$ otherwise.

2.1 Scheduling and Latency

That of packet scheduling for wired networks has been a flourishing literature stream during the last two decades (see, e.g., [7]). Some (actually, a minority) of the devised scheduling algorithms have been implemented in commercial routers (e.g., [5,12]), or made their way into the codebase of open-source operating systems (e.g., [5]). There are three main expressions for latency, to which we associate names for ease of notation. A good survey on the subject can be found in [8]. Call L the Maximum Transmit Unit (MTU) in the network (assumed to be equal at all links and for all flows for notational simplicity, although the model can be easily generalized). The following latency expressions can be defined.

1. Strictly rate-proportional (SRP) latency (PGPS [4], WF^{2}Q [15], Virtual Clock [13], etc.):

$$\theta_{(x,y),(i,e)} = \frac{L}{R_{(x,y),(i,e)}} + \frac{L}{W_{(x,y)}} \tag{3}$$

2. weakly rate proportional (WRP) latency (e.g., Self-Clocked Fair Queuing, [10]):

$$\theta_{(x,y),(i,e)} = \frac{L}{R_{(x,y),(i,e)}} + \left(n_{(x,y)} - 1\right) \cdot \frac{L}{W_{(x,y)}}, \tag{4}$$

where $n_{(x,y)}$ is the number of flows traversing link (x, y).

3. frame-based (FB) latency (e.g. DRR [5], [11][2]):

$$\theta_{(x,y),(i,e)} = \frac{L}{W_{(x,y)}} \left[\left(W_{(x,y)} - R_{(x,y),(i,e)}\right) \cdot \left(\frac{1}{\min\limits_{(a,b):(x,y)\in P_{(a,b)}} \left\{R_{(x,y),(a,b)}\right\}} + \frac{1}{R_{(x,y),(i,e)}} \right) + n_{(x,y)} \right] \tag{5}$$

Other frame-based schedulers have recently been derived from DRR (e.g. EBDRR [13], ALiQueM [11] etc.), and improve on its latency by dividing some of the above addenda by a constant term. As the purpose of this paper is to investigate resource allocation under delay constraints (rather than surveying all possible schedulers) we leave these minor generalization to the interested reader.

In all three cases, increasing the rate of a flow decreases its latency, although the effectiveness of such a tuning clearly decreases when we move from category 1) to 3). In fact, 2) contains an $n_{(x,y)}$ term, whereas 3) includes the latter and a minimum rate at the denominator, which cannot be modified by increasing $R_{(x,y),(i,e)}$.

2.2 Path Computation Algorithms

The path computation algorithms that we consider in this work are the following:

1. Constrained Multicommodity Shortest Path First (CM-SPF): assuming that links are characterized by capacities and have unitary weights, it computes the shortest path from a source to a destination having at least the required capacity. All requests are considered jointly, and the result is the set of paths having the minimum total number of hops.
2. Constrained Shortest Path First (C-SPF): the same as the previous one, but with sequential computations, so that the outcome depends on the order in which path requests are considered.
3. Widest-Shortest Path First (WPF): the same as 2), with the difference that links have weights which are inversely proportional to the residual capacity on each link.
4. Maximum Maxflow (MM, [27]): for each request, the path that yields the maximum (weighted) sum of the maxflows between any source and destination pair is

[2] The latency expression reported here can be worked out via straightforward algebraic manipulations from the one reported in [11]. The one in [8] is instead an overrated bound.

selected. This way, the ability to route future requests between a source and destination is maximized, though generally at the expenses of having longer paths. MM was proved in [27] to be NP-hard. Authors propose a heuristic algorithm (called Minimum Interference Routing, MIRA) to approximate the MM solution in polynomial time.

3 Optimal Resource Allocation

A joint routing and resource allocation problem can be formulated as follows:

Joint Routing and Resource Allocation Problem (JRRA)
For each flow (i,e), i) compute a path $P_{(i,e)}$ and ii) reserve a rate on all the links in $P_{(i,e)}$ (subject to constraints (1)) so that $D_{P_{(i,e)}} \leq \delta_{(i,e)}$, if it is possible to do so.

The above one is a *feasibility* problem, claimed to be NP-hard in [19] (where, however, it is formulated assuming that $R_{(x,y),(i,e)} = R_{(i,e)}$, although we do not believe that relaxing the above constraint is going to make the problem any easier). It can be turned into an *optimization* problem once a suitable objective function to be minimized or maximized is identified. Several such functions can be envisaged, such as:

1. maximizing the minimum slack with respect to the delay bound. A non-positive objective means that all slacks are non negative, i.e. that all delay bound inequalities are verified, hence the solution is feasible. It has been shown in [6], although in a slightly different context, that this formulation leads to robust schedules, which can easily tolerate uncertainties in the parameters. On the cons side, such an approach tends to allocate rates too liberally, thus depleting the resources. Even though we deal with a static environment, where all requests are known a priori, it is intuitively reasonable to try to minimize the amount of reserved rate, so as to leave the maximum possible room for future requests or cope with parameter uncertainties.
2. Maximizing the total unreserved capacity in the network, i.e. the sum of the slacks in (1), again having the delay bounds as constraints. This allows for a higher number of future requests to be considered. If necessary, a cost $c_{(x,y)}$ can be statically associated to a capacity unit on each link, so as to reflect their relative importance.
3. Maximizing the sum of the maxflows between each source/destination pairs.

As already anticipated, in this paper we do not solve the above problem, but instead mark a first step in that direction by researching the utility of global resource minimization techniques. We reformulate the resource allocation sub-problem as follows:

Global Resource Allocation Problem (GRA)
Given a set of paths $P_{(i,e)}$ for all flows $(i,e) \in F$, compute the vector of the allocated rates along a path $P_{(i,e)}$, $\mathbf{R} = \left\{ R_{(x,y)}, (x,y) \in P_{(i,e)} \right\}$, (subject to constraints (1)) so that i) $D_{P_{(i,e)}} \leq \delta_{(i,e)}$, and ii) the sum of the allocated rates is minimum, if it is possible to do so.

The GRA problem assumes that paths have been precomputed, using any of the techniques described in Section 2.2. Given the flow routes, leaky-bucket profiles and deadline requirements, the GRA problem can be formulated as the following optimization problem:

$$\begin{aligned} \min \quad & \sum_{(i,e)} \sum_{(x,y)\in P_{(i,e)}} R_{(x,y),(i,e)} \\ \text{s.t.:} \quad & D_{P_{(i,e)}} \le \delta_{(i,e)} && \forall (i,e) \in F \\ & R_{\min}^{(i,e)} \le R_{(x,y),(i,e)} && \forall (x,y) \in P_{(i,e)}, \forall (i,e) \in F \end{aligned}$$

We select the sum of the allocated rates as the objective to be minimized, without considering link costs. The latter can obviously be added back if necessary, without changing the nature of the problem. The 1st constraint ensures that all the flows meet their deadline delay requirements on their respective paths. The $D_{P_{(i,e)}}$ delay expression can be defined using any of the three latencies defined in Section 2.1. Latencies (3) and (4) are convex, since they involve summations of convex function, while (5) is not defined. Hence the resulting GRA problems are convex non-linear optimization problems for latencies (3) and (4), non-convex non-linear optimization problems for latency in (5) respectively. The non linear constraints of the *convex* formulations contain hyperbolic constraints that can be reformulated as Second Order Cone Programming (SOCP) programming [32] and solved using a solver for quadratically constrained programs. In this case, interior point methods can be used, which complete in polynomial time and are generally very fast. In the non convex case, instead, global optimization is required, which is considerably more complex. We solved these problems using general purpose solvers such as CPLEX [28] and BARON [29].

To deal with the min operator in (2), additional variables $R_{\min}^{(i,e)}$, representing the minimum rate on a path $P_{(i,e)}$, are required: the 2nd constraints couple these variables with the link rates. Since the objective function is a minimization problem of the link rates, these variables will be assigned the lowest rate that guarantees that no deadline is violated, according to the 1st constraint. If the latency defined in (5) is used instead, additional variables $R_{\min}^{(x,y)}$, representing the minimum rate at a *link* (x,y) (among those allocated to the flows traversing that link), are also required together with the following constraints:

$$R_{\min}^{(x,y)} \le R_{(x,y),(i,e)} \quad \forall (x,y) \in P_{(i,e)}, \forall (i,e) \in F$$

The above problem has $O\left(|F| \cdot |E|\right)$ variables and constraints.

As already observed in [20], allocating the *same rate* at all links for a flow, i.e. $R_{(x,y),} = R_{(i,e)}$, $(x,y) \in P_{(i,e)}$, may lead to suboptimal rate allocations. However, under that assumption the allocation problem can be solved analytically, once the path and the schedulers at all links are known. Hence, we will use this approach (henceforth referred to as *equal rate allocation, ERA*) as a comparison, to test how much can be harvested by using global minimization. The minimum required rate on a single path can be computed by solving (2) with respect to the rate, under the assumption that

$D_{P_{(i,e)}} = \delta_{(i,e)}$. Such allocation is in fact exploited by RSVP to compute the rates to allocate at each link for IntServ guaranteed-rate connections [1]. We remark that, since this allocation is done path-wise, then (1) may be violated even if a feasible (e.g. globally computed) rate allocation exists.

For strictly and weakly rate-proportional latencies (3) and (4), the alert reader can check that the ERA solutions are the following:

$$R_{(i,e)} = \max\left\{\rho_{(i,e)}, \frac{\left|P_{(i,e)}\right| \cdot L + \sigma_{(i,e)}}{\delta_{(i,e)} - \sum_{(x,y)\in P_{(i,e)}} \left[\frac{L}{W_{(x,y)}} + tp_{(x,y)} + td_x\right]}\right\}, \tag{6}$$

$$R_{(i,e)} = \max\left(\rho_{(i,e)}, \frac{\left|P_{(i,e)}\right| \cdot L + \sigma_{(i,e)}}{\delta_{(i,e)} - \sum_{(x,y)\in P_{(i,e)}} \left[\left(n_{(x,y)} - 1\right) \cdot \frac{L}{W_{(x,y)}} + tp_{(x,y)} + td_x\right]}\right), \tag{7}$$

if the denominator is positive (which is a necessary condition for the problem to have a solution at all to the allocation problem, whether global or path-wise). $\left|P_{(i,e)}\right|$ denotes the number of hops in path $P_{(i,e)}$.

On the other hand, for frame-based latency (5), computing the same required rate $R_{(i,e)}$ is considerably more involved, and requires global optimization, due to the *min* term in that expression. In fact, since the latency at a link depend on the minimum rate allocated at a link (possibly to some other flow), all flows traversing the same link should be considered simultaneously in order to determine the latency of each one. However, we can do a reasonable approximation by considering that, by definition, it is:

$$R_{(i,e)} \geq \min_{\substack{(a,b):(x,y)\in P_{(a,b)} \\ (x,y)\in P_{(i,e)}}} \left\{R_{(x,y),(a,b)}\right\}. \tag{8}$$

Hence, we can obtain a rate $R_{(i,e)}$ that leads to feasible delays (if that rate is indeed available at all links), by merging (5) and (2) and assuming that equality holds in (8):

$$R_{(i,e)} = \max\left(\rho_{(i,e)}, \frac{2\left|P_{(i,e)}\right| \cdot L + \sigma_{(i,e)}}{\delta_{(i,e)} - \sum_{(x,y)\in P_{(i,e)}} \left[\left(n_{(x,y)} - 2\right) \cdot \frac{L}{W_{(x,y)}} + tp_{(x,y)} + td_x\right]}\right) \tag{9}$$

Note that, with (9), it is $D_{P_{(i,e)}} \leq \delta_{(i,e)}$, and inequality may actually hold, i.e. the rate can be overprovisioned with respect to the one strictly required to meet the deadline. The alert reader can easily check that the amount of required rate to meet a given deadline increases with the latency model, from (6) to (7) to (9).

4 Numerical Results

In order to prove the effectiveness of the proposed approach, we performed simulations on a sample network, shown in Fig. 1. Links are bidirectional, with either $W_{(x,y)}$=35Mbps or $W_{(x,y)}$=75Mbps speed. We assume that $tp_{(x,y)} = 0$, $td_x = 0$ and $C_{(x,y)} = W_{(x,y)}$ for simplicity. Furthermore, it is $L = 1.5$ kB.

We generate 96 flows, between random pairs of nodes. All the flows have homogeneous QoS requirements with $\rho = 1$ Mbps, $\sigma = 12$ kB and $\delta = 15$ ms. We use all the path computation algorithms mentioned in Section 2.2, and all the latency models of Section 2.1, and we compute both the optimal solution of the GRA in the above settings and the ERA (6), (7) and (9).

ERA is known to be optimal when link capacities are unbounded [25]. While this seems to imply that at *low loads*, i.e., when capacity bounds are not active constraints, an equal allocation is the optimal one, we show that even at low utilizations this approach may yield infeasible solutions. For instance, in our case the total available bandwidth is 2530Mbps, and the overall rate demand is 96Mbps, which is 3.8%. Using ERA, the *average* link utilization is between 20% and 24%, depending on the path computation scheme and latency adopted. Yet, there are links whose capacity is exceeded, thus leading to an unfeasible solution, whereas a feasible one can be found by optimally solving the GRA.

Consider for instance the case of C-SPF and strictly rate-proportional latencies. In this case, the capacity reserved by ERA at link (6,3) exceeds the available one by 6.28%, and accordingly bounds cannot be guaranteed to all flows traversing that link. On the other hand, the optimal solution to the GRA problem is feasible, as it exploits the ability to assign different rates on different links of the same flow path. For instance, the optimal assignment for flow (8,0) – which traverses link (6,3) – is the following:

Link (x,y)	$R_{(x,y),(8,0)}$ **(Mbps)**	**Link utilization**
(8,5)	3.90	1
(5,6)	6.91	1
(6,3)	1.14	1
(3,2)	5.52	1
(2,0)	10.75	0.97

The ERA rate would instead have been 1.16 Mbps, i.e. slightly more (although critically so) than the one allocated by GRA on the critical link. Note that the lack of rate at the critical link is made up for by allocating more resources on uncongested

peripheral links, so that the allocated rate is highly inhomogeneous. For flows traversing unloaded links the rate provided using the analytical framework is the optimal solution as shown for the rate assignment of flow (12, 4):

Link (x,y)	$R_{(x,y),(12,4)}$	Link utilization
(7,4)	1.16	0.11
(13,7)	1.16	0.14
(12,13)	1.16	0.56

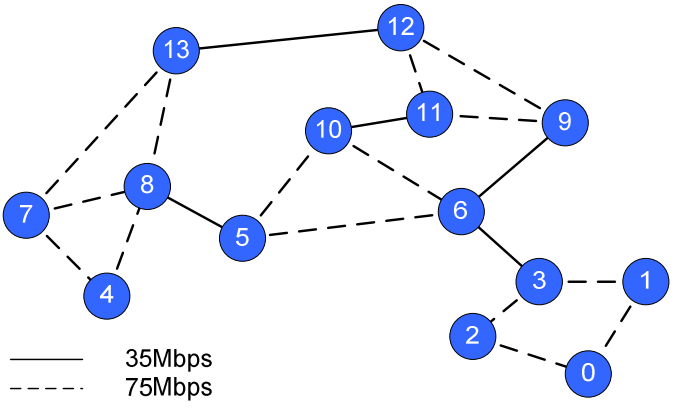


Fig. 1. Sample network for numerical analysis

On a more general note, we can also derive some considerations from aggregated utilization data. Fig. 2 shows the number of oversubscribed links for each path computation, rate-proportional latency model and resource allocation scheme. Note that, for frame-based latency, we only report one example, using C-SPF as path computation (due to the larger overhead of solving non-convex problems). Furthermore, the scenarios with FB latency are with a reduced number of flows (80 instead of 96), otherwise the problem is not feasible. As the figure shows, GRA always finds feasible solutions, whereas ERA is always unfeasible. The oversubscribed links with ERA are (6,3), whereas (3,6) is oversubscribed under WRP and FB latency, and (5,8), only with CM-SPF and WRP latency. Data related to the MM path computation scheme were not reported, as no feasible solution to the GRA can be found for the scenario under consideration in that sense.

Fig. 3 shows the average utilization for the links under the various configurations. The figure shows that, while the latency model does not play a big difference with ERA (few percentage points), it does so under GRA, where WRP latency requires almost double as many resources to be allocated than SRP latency, in order to maintain feasible delays. The figure also shows that negligible differences exist among the path computation schemes (except for the case of MM, as already explained). The main difference between MM and the rest is that the former computes considerably longer paths, so that the amount of resources required to maintain feasible delays is multiplied by a higher path length.

Fig. 4 and 5 show the utilization per link under WRP latency and CM-SPF, with both ERA and GRA. Consecutive pairs of links are the forward and reverse direction of the link showed in the horizontal axis. As already explained, ERA oversubscribes three links, whereas GRA does not. What is remarkable in that figure is that the amount of resources that are required in order to keep the delay bounds within s bounded is indeed taxing, with eight links fully booked. This seems to suggest that there is room for improving the efficiency of the allocation by *jointly* solving routing and resource allocation.

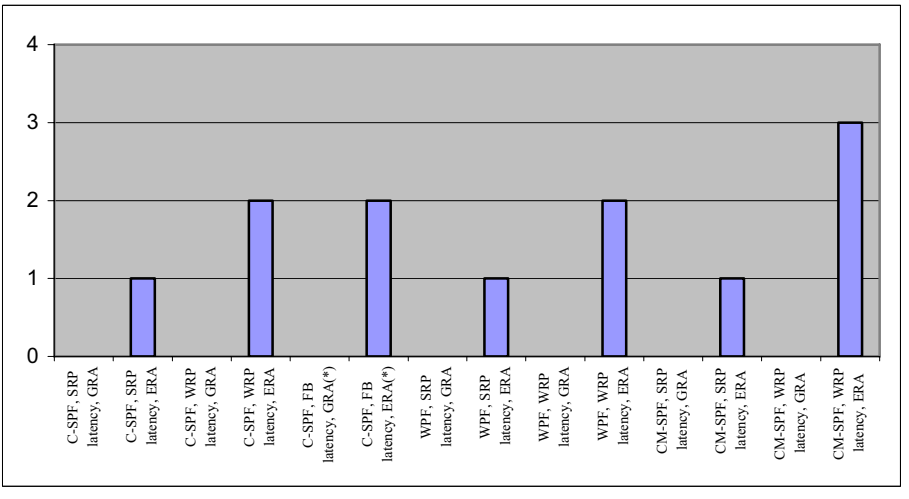


Fig. 2. Number of oversubscribed links

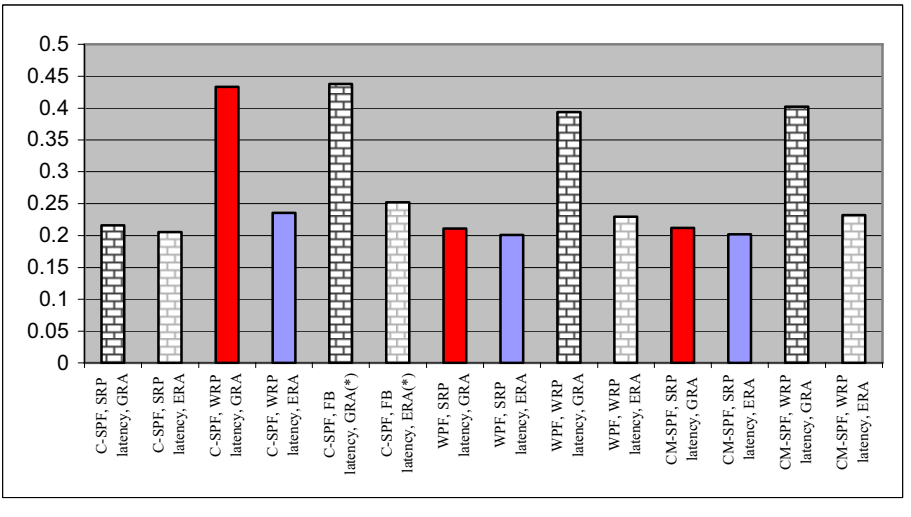


Fig. 3. Average utilization for the links

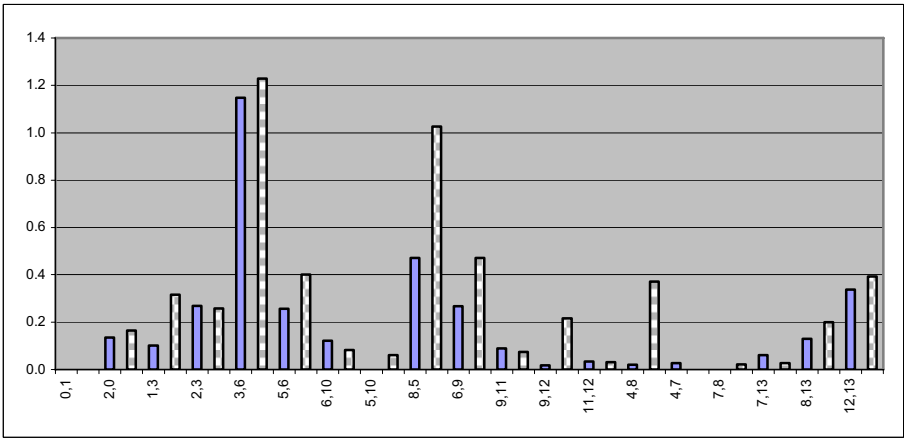


Fig. 4. Link utilization – CM-SPF, with WRP latency, ERA

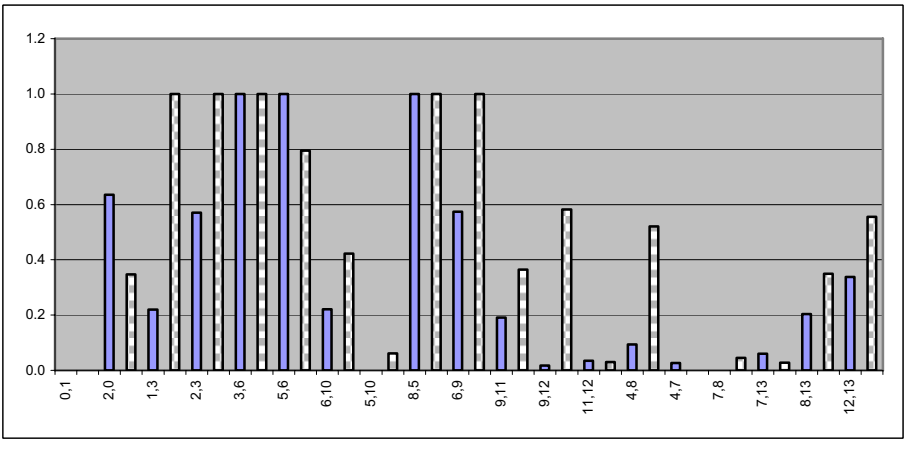


Fig. 5. Link utilization – CM-SPF, with WRP latency, GRA

5 Conclusions and Future Work

This paper explored the space for global optimization in resource allocation for guaranteed-delay traffic engineering. We formulated and solved the problem under different latency models, showing that the adopted latency does indeed make a difference as far as resource consumption is concerned. Our results on a case-study network show that, even at surprisingly low average loads, using global optimization can help feasible schedules to be computed, whereas local resource allocation schemes would fail to do so.

This work marks the first exploration in a rather broad research field, and, as such, calls for extension along several directions. The first one is to derive a joint

framework for path computation and resource allocation, taking into account real-time constraint. Second, we are currently considering including stochastic network calculus [30] in the framework. This would allow us to relax the assumption of deterministic worst-case delay, while still retaining quantifiable probabilistic guarantees, although possibly at the expenses of additional hypotheses on the traffic. This, in turn, would allow us to capitalize on statistical multiplexing, possibly increasing the amount of carried traffic within the network. Third, while the *pipe* (i.e., point-to-point) path model is probably the most widely used in TE practices, the *funnel* and *hose* models (i.e., multipoint-to-point and point-to-multipoint respectively) can also be used. In those cases, resource allocation is done on a per-tree basis, and delay bounds have different formulations [31]. Analyzing these networks is part of the ongoing work.

References

1. Braden, R., Clark, D., Shenker, S.: Integrated Services in the Internet Architecture: an Overview. RFC 1633, The Internet Society (June 1994)
2. Rosen, E., Viswanathan, A., Callon, R.: Multi Protocol Label Switching Architecture. IETF RFC 3031 (2001)
3. Le Boudec, J.-Y., Thiran, P.: Network Calculus. LNCS, vol. 2050. Springer, Heidelberg (2001)
4. Parekh, K., Gallager, R.G.: A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: the Single Node Case. IEEE/ACM Trans. on Networking 1, 344–357 (1993)
5. Shreedhar, M., Varghese, G.: Efficient Fair Queueing Using Deficit Round Robin. IEEE/ACM Trans. on Networking 4, 375–385 (1996)
6. Cappanera, P., Lenzini, L., Lori, A., Stea, G., Vaglini, G.: Optimal Link Scheduling for Real-time Traffic in Wireless Mesh Networks in both Per-flow and Per-path Frameworks. In: Proceedings of WoWMoM 2010, Montreal, CA (June 2010)
7. Zhang, H.: Service Disciplines for Guaranteed Performance Service in Packet-Switching Networks. Proceedings of the IEEE 83(10), 1374–1396 (1995)
8. Stiliadis, D., Varma, A.: Latency-Rate Servers: A General Model for Analysis of Traffic Scheduling Algorithms. IEEE Transaction on Networking 6(5), 675–689 (1998)
9. Goyal, P., Vin, H.M., Cheng, H.: Start-Time Fair Queueing: A Scheduling Algorithm for Integrated Services Packet Switching Networks. IEEE/ACM Trans. on Networking 5(5), 690–704 (1997)
10. Golestani, S.J.: A Self-Clocked Fair Queueing Scheme for Broadband Applications. In: Proc. of IEEE INFOCOM 1994, Toronto, Canada, pp. 636–646 (June 1994)
11. Lenzini, L., Mingozzi, E., Stea, G.: Tradeoffs between Low Complexity, Low Latency and Fairness with Deficit Round Robin Schedulers. IEEE/ACM Transactions on Networking, 681–693 (August 2004)
12. Lenzini, L., Mingozzi, E., Stea, G.: Performance Analysis of Modified Deficit Round Robin Schedulers. IOS Journal of High-Speed Networks 16(4), 399–422 (2007)
13. Lenzini, L., Mingozzi, E., Stea, G.: Eligibility-Based Round Robin for Fair and Efficient Packet Scheduling in Interconnection Networks. IEEE Transactions on Parallel and Distributed Systems, 254–266 (March 2004)
14. Zhang, L.: Virtual clock: a new traffic control algorithm for packet switching networks. ACM SIGCOMM Computer Communication Review 20(4), 19–29 (1990)

15. Bennett, J.C.R., Zhang, H.: WF2Q: Worst-case Fair Weighted Fair Queueing. In: INFOCOM 1996 (March 1996)
16. Osborne, E., Simha, A.: Traffic Engineering with MPLS. Cisco Press (July 17, 2002)
17. Awduche, D., Malcolm, J., Agogbua, D., O'Dell, M., McManus, J.: Requirements for Traffic Engineering Over MPLS. RFC 2702 (September 1999)
18. Ma, Q., Steenkiste, P.: Quality-of-Service Routing for Traffic with Performance Guarantees. IFIP Networking (1997)
19. Orda, A.: Routing with End to End QoS Guarantees in Broadband Networks. IEEE/ACM Transactions on Networking (1999)
20. Diwan, A., Kuri, J., Kumar, A.: Optimal per-Node Rate Allocation to provide per-Flow End-to-End Delay Guarantees in a Network of Routers supporting Guaranteed Service Class. In: ICC 2002, New York, USA, April 28 - May 2, vol. 2, pp. 1112–1117 (2002)
21. Misra, S., Xue, G., Yang, D.: Polynomial Time Approximations for Multi-Path Routing with Bandwidth and Delay Constraints. In: INFOCOM 2009 (2009)
22. Lorenz, D.H., Orda, A.: Optimal Partition of QoS Requirements on Unicast Paths and Multicast Trees. IEEE/ACM Transactions on Networking 10(1), 102–114 (2002)
23. Yang, W.-L.: Optimal and heuristic algorithms for quality-of-service routing with multiple constraints. Performance Evaluation 57, 261–278 (2004)
24. Elsayed, K.M.F.: A Framework for End-to-End Deterministic-Delay Service Provisioning in Multiservice Packet Networks. IEEE Transactions on Multimedia 7(3), 563–571 (2005)
25. Saad, M., Leon-Garcia, A., Yu, W.: Optimal Network Rate Allocation under End-to-End Quality-of-Service Requirements. IEEE Transactions on Network and Service Management 4(3) (December 2007)
26. Chen, S., Nahrstedt, K.: An Overview of Quality-of-Service Routing for the Next Generation High-Speed Networks: Problems and Solutions. IEEE Network Mag. 12(6), 64–79 (1998)
27. Kar, K., Kodialam, M., Lakshman, T.V.: Minimum Interference Routing of Bandwidth Guaranteed Tunnels with MPLS Traffic Engineering Applications. IEEE Journal on Selected Areas in Communications 18, 2566–2579 (2000)
28. ILOG CPLEX software, `http://www.ilog.com`
29. BARON solver (available freely via the NEOS server), `http://archimedes.cheme.cmu.edu/baron/baron.html`
30. Jiang, Y., Liu, Y.: Stochastic Network Calculus. Springer, Heidelberg (2008)
31. Lenzini, L., Martorini, L., Mingozzi, E., Stea, G.: Tight End-to-end Per-flow Delay Bounds in FIFO Multiplexing Sink-tree Networks. Performance Evaluation 63, 956–987 (2006)
32. Lobo, M., Vandenberghe, L., Boyd, S., Lebret, H.: Applications of Second-Order Cone Programming. Linear Algebra and its Applications 284, 193–228 (1998)

Partially Synchronizing Periodic Flows with Offsets Improves Worst-Case End-to-End Delay Analysis of Switched Ethernet

Xiaoting Li, Jean-Luc Scharbarg, and Christian Fraboul

Université de Toulouse - IRIT/ENSEEIHT/INPT - Toulouse, France
{Xiaoting.Li,Jean-Luc.Scharbarg,Christian.Fraboul}@enseeiht.fr

Abstract. A switched Ethernet network needs a worst-case delay analysis to prove that real-time applications can be correctly implemented on it. Existing approaches for upper bounding end-to-end delay assume that there is either no-synchronization or a global synchronization of flows which needs a global clock. Conversely, flows emitted by the same end-system can be easily synchronized using the existing local clock. This paper explains how the partial synchronization of periodic flows can be integrated in the worst-case delay analysis provided by Network Calculus approach. End-to-end delays computed on switched Ethernet configurations show that the proposed method clearly reduces the upper bounds of end-to-end delay.

1 Introduction

Full duplex switched Ethernet network eliminates the CSMA/CD indeterminism but shifts the issue at the switch level where temporary congestion on an output port can lead to variable delays. Thus, using such a technology in a real-time context implies a worst-case delay analysis. Indeed, it is important to demonstrate that an upper bound can be determined for end-to-end (ETE) communication delay on such network. This guarantee can be provided by different approaches [1,2,3,4]. These approaches consider that all the flows are not synchronized, which leads to pessimistic upper bounds for scheduled periodic flows. A global synchronization of the flows induces an extra cost (global clock) which cannot be neglected. However partial synchronization of flows transmitted by the same source node can be easily done according to the local clock of this system.

The aim of this paper is to evaluate the improvement brought by partial synchronization of flows on worst-case delay analysis. To achieve that, an approach based on the Network Calculus is developed so as to introduce the scheduling of periodic flows in the worst-case analysis. This scheduling can be modeled by associating offsets to different periodic flows.

The paper is organized as follows. Section 2 presents the network model and the influence of the partial synchronization brought by each node on the flows. The worst-case analysis on ETE delay based on Network Calculus approach with

T. Margaria and B. Steffen (Eds.): ISoLA 2010, Part I, LNCS 6415, pp. 228–242, 2010.

the partial synchronization is developed in Section 3. In Section 4, two topologies of switched Ethernet are evaluated and the improvements in terms of computed ETE delay upper bounds are presented. Section 5 concludes and discusses the direction for future work.

2 End-to-End Delay Analysis

2.1 Network and Traffic Model

This work focuses on the worst-case delay analysis of a switched Ethernet network, composed of multiple nodes interconnected by switches using a store and forward pattern. All the links are point-to-point bidirectional full-duplex, in order to guarantee a collision-free system. All the nodes and switches support the first-in-first-out (FIFO) queuing.

The incoming flows are strictly periodic. Each *periodic flow* (PF) sends frames from the source node to the destination nodes (multicast case included). One PF consists of one or several paths which are statically defined. A PF pf_m is characterized by its period P_m and the lower and upper bounded frame sizes L^m_{min} and L^m_{max}.

The end-to-end delay of a frame following a path consists of the transmission delay on links, the switch-dependent delay caused by an upper bounded technological latency and the waiting delay caused by competition with other frames in the output buffers. The upper bounds of the first two parts are fixed and depend on the transmission rate of links, the number of switches in the path, the size of frames and the technological latency of each switch. The last part is variable because it depends on the load of the output ports of each crossed switch at the time when the considered frame reaches it. The ETE delay upper bounds rely on the upper bounded third part. Meanwhile, the minimum ETE delay of a frame is the sum of the first two parts.

The set of flows emitted by a given source node N_i are scheduled by N_i. Conversely, the different nodes are totally asynchronous. It leads to a partial synchronization of flows, which affects the last part of the ETE delay. This influence is studied in the next paragraph.

2.2 Influence of Partial Synchronization

The small example depicted in Figure 1 is considered to illustrate the influence of the partial synchronization on flows. The parameters of each PF of the configuration in Figure 1 are given in Table 1.

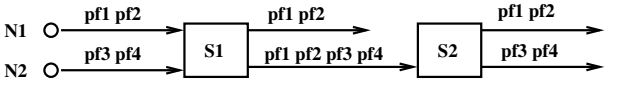


Fig. 1. Example of a switched Ethernet

Table 1. Configuration of sample switched Ethernet

PF	$P\ (ms)$	$L_{max}\ (Byte)$	$L_{min}\ (Byte)$
pf_1	8	500	125
pf_2	4	750	125
pf_3	2	500	250
pf_4	16	1000	500

Let's focus on the temporal distance between a frame f_1 of pf_1 and a frame f_2 of pf_2 at N_1 in Figure 1. Without partial synchronization at N_1, there is no constraint on pf_1 and pf_2. It means that f_1 and f_2 can arrive at the same time. Then one frame can be delayed by the other one. This situation is depicted as A in Figure 2, where $a_{f_m}^{N_i}$ is the arrival time of frame f_m at N_i. However, each source node schedules its PFs using its local clock. This scheduling can be modeled by the offset assigned to each PF.

Definition 1. *The **Definitive Offset (DO)** of a given PF pf_m is the release time of its first frame at its source node N_i, denoted $O_{d,m}^{N_i}$.*

Obviously, the *Definitive Offset* is settled by the scheduling of the corresponding node. For pf_1 and pf_2 in Figure 1, $O_{d,1}^{N_1} = a_{f_1}^{N_1}$ and $O_{d,2}^{N_1} = a_{f_2}^{N_1}$, and the *DOs* of pf_1 and pf_2 are assigned to $O_{d,1}^{N_1} = 0\mu s$, $O_{d,2}^{N_1} = 1500\mu s$ (B in Figure 2). The *DOs* can bring some temporal separations between frame transmissions. This temporal separation can be modeled by a relative offset.

Definition 2. *The **Relative Offset (RO)** is defined based on a benchmark PF (bPF). The RO of another PF pf_n is the minimum possible time interval between any frame f_m of the bPF pf_m and the first frame f_n of pf_n transmitted after f_m.*

The *RO* of pf_n at N_i when pf_m is the bPF is denoted $O_{r,m,n}^{N_i}$. In the general case, a source node N_i schedules x PFs: $pf_1, ..., pf_x$. The *Relative Offsets* have to be computed considering each PF among $pf_1, ..., pf_x$ as the benchmark PF. Defining a set $\mathcal{J} = \{1, ..., x\}$, then each $O_{r,m,n}^{N_i}$ with $m \in \mathcal{J}, n \in \mathcal{J} \setminus m$ needs to be computed. Considering N_1 in Figure 1, $O_{r,1,2}^{N_1}$ and $O_{r,2,1}^{N_1}$ have to be computed.

For a set of periodic flows scheduled by a given source node N_i, the computation of the *ROs* $O_{r,m,n}^{N_i}$ for any pf_m and pf_n is straightforward. Let's consider that $O_{r,m,n}^{N_i}(k,l)$ is the time interval between the k^{th} frame of pf_m and the l^{th} frame of pf_n. Then $O_{r,m,n}^{N_i}(k,l) = (O_{d,n}^{N_i} + P_n \cdot l) - (O_{d,m}^{N_i} + P_m \cdot k)$. The *Relative Offset* $O_{r,m,n}^{N_i}$ is the smallest non negative value of $O_{r,m,n}^{N_i}(k,l)$ for any $k \geq 0, l \geq 0$:

$$O_{r,m,n}^{N_i} = \min_{\substack{k \geq 0, l \geq 0 \\ O_{r,m,n}^{N_i}(k,l) \geq 0}} (O_{r,m,n}^{N_i}(k,l)) \tag{1}$$

For N_1 in Figure 1, $O_{r,1,2}^{N_1}(k,l) = (1500 + 4000 \times l) - (0 + 8000 \times k)$, then $O_{r,1,2}^{N_1} = 1500\ \mu s$ when $k = 0, l = 0$. Similarly $O_{r,2,1}^{N_1} = 2500\ \mu s$ (B in Figure 2).

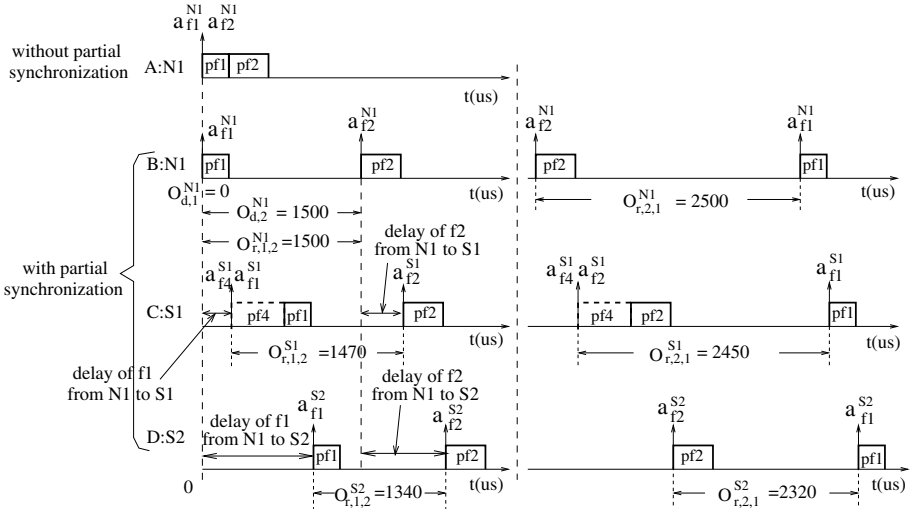


Fig. 2. Relative offsets of pf_1 and pf_2

For two flows pf_m and pf_n scheduled by the same source node N_i and sharing the path $\{N_i, ..., S_k\}$, the *ROs* have to be computed on every switch output port shared by pf_m and pf_n. Let's consider the general case depicted in Figure 3. The frame f_m of pf_m and the frame f_n of pf_n are separated by the *RO* of pf_n at N_i when pf_m is bPF: $O^{N_i}_{r,m,n}$. The *RO* of pf_n at switch S_k when pf_m is the bPF is denoted $O^{S_k}_{r,m,n}$. In the scenario in Figure 4, the delay of f_m from N_i to S_k is bigger than that of f_n because f_m is delayed by more competing flows between N_i and S_k. In this case, the time interval between f_m and f_n at S_k becomes smaller than $O^{N_i}_{r,m,n}$. $O^{S_k}_{r,m,n}$ is the smallest time interval between f_m and f_n at S_k, which means that it may not be equal to $O^{N_i}_{r,m,n}$ as illustrated in Figure 4. Hence, it is necessary to compute *Relative Offset* at each switch output port.

In order to achieve the minimum time interval $O^{S_k}_{r,m,n}$ between f_m and f_n, the delay of f_m from N_i to S_k should be maximized while that of f_n should be minimized. The maximum delay of frame f_m from the source node N_i to the input of the switch S_k is denoted $D^{S_k}_{max,m}$, and occurs when f_m is delayed by all possible competing flows at each crossed switch between N_i and S_k. Let's consider pf_1 in Figure 1 as the bPF. Its delay between N_1 and S_2 should be

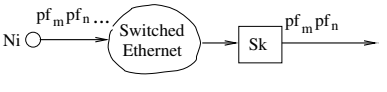


Fig. 3. General case

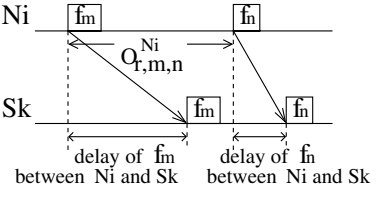


Fig. 4. One possible scenario

maximized in order to compute $O^{S_2}_{r,1,2}$. At N_1, there is no competing flows for f_1 because of the offsets and then $D^{S_1}_{max,1} = 40\ \mu s$ as depicted in B in Figure 2. At the output of S_1, pf_1 and pf_2 compete with pf_3 and pf_4. pf_3 and pf_4 are scheduled at N_2 and separated by the offsets, therefore only one of them can delay pf_1 at S_1. Since L^4_{max} of pf_4 is larger than L^3_{max} of pf_3, the worst case for pf_1 occurs when the frames of pf_1 and pf_4 arrive at S_1 at the same time (C in Figure 2) and $D^{S_2}_{max,1} = 196\ \mu s$. Section 3 will present an approach for the computation of this worst case delay.

The minimum delay of frame f_n from the source node N_i to the input of the switch S_k is denoted $D^{S_k}_{min,n}$ and occurs when f_n is not delayed by any competing flows. Provided that the transmission rate of switched Ethernet is R, there are h segments between N_i and S_k, and the technological latency is T_l, the minimum delay is given by:

$$D^{S_k}_{min,n} = (\frac{L^n_{min}}{R} \times h) + T_l \times (h-1) \tag{2}$$

On the example in Figure 1, $D^{S_1}_{min,2} = 10\ \mu s$ and $D^{S_2}_{min,2} = 36\ \mu s$.

To summarize, $O^{S_k}_{r,m,n}$ is given by:

$$O^{S_k}_{r,m,n} = O^{N_i}_{r,m,n} + D^{S_k}_{min,n} - D^{S_k}_{max,m} \tag{3}$$

Then on the example in Figure 1, by formula 3, we have:

$$O^{S_1}_{r,1,2} = O^{N_1}_{r,1,2} + D^{S_1}_{min,2} - D^{S_1}_{max,1} = 1500 + 10 - 40 = 1470\ \mu s$$

$$O^{S_2}_{r,1,2} = O^{N_1}_{r,1,2} + D^{S_2}_{min,2} - D^{S_2}_{max,1} = 1500 + 36 - 196 = 1340\ \mu s$$

A similar computation can be done with pf_2 as the bPF. The results are depicted in C and D in Figure 2.

The *ROs* have an influence on the ETE delays of flows. The next section shows how these *ROs* can be integrated in the worst-case delay analysis of a switched Ethernet network.

3 Worst-Case Delay Analysis with Partial Synchronization

3.1 Basic Network Calculus Approach for ETE Delay Analysis

The Network Calculus has been introduced for calculating network worst-case delay and it has been applied to switched Ethernet [5,6,7]. The Network Calculus is briefly summarized in this context.

For a network element, its input function $F(t)$ is defined as the amount of traffic that arrives at the element over time, while its output function $F^{'}(t)$ is defined as the amount of traffic that departs from the element over time. Based

on min-plus algebra, the convolution and deconvolution of two functions f and g are defined respectively as follows:

$$(f \otimes g)(t) = \inf_{0 \le s \le t} \{f(t-s) + g(s)\}, \ (f \oslash g)(t) = \sup_{u \ge 0} \{f(t+u) - g(u)\}$$

A flow represented by its input function $F(t)$ is constrained by an *arrival curve* $\alpha(t)$ if and only if $F(t) \le (F \otimes \alpha)(t)$. In the context of switched Ethernet, the PF is constrained by a leaky bucket model $\alpha(t) = rt + b$, with the burst tolerance $b = L_{max}$ and the peak rate $r = \frac{L_{max}}{P}$. Let's consider the example in Figure 1. For pf_1, L^1_{max} is $500 \times 8 = 4000$ bits and P_1 is 8000 μs. Consequently $r = \frac{4000}{8000} = 0.5$ and $\alpha_{pf_1}(t) = 0.5t + 4000$. Similarly, for pf_2, $\alpha_{pf_2}(t) = 1.5t + 6000$. α_{pf_1} and α_{pf_2} are depicted in Figure 5.

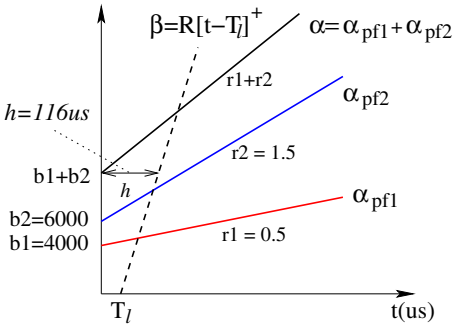


Fig. 5. Illustration of Network Calculus approach

A server offers to the flow with input function $F(t)$ a service curve $\beta(t)$ if and only if $F^{'}(t) \ge (F \otimes \beta)(t)$, where the output function $F^{'}(t)$ is constrained by $\alpha^{'}(t) = (\alpha \oslash \beta)(t)$. The service curve provided by each output port of switch is $\beta(t) = R[t - T_l]^+$ where R is the transmission rate of link, T_l is the upper bounded technological latency and $[x]^+ = max(0, x)$. Let's consider that the network works at $100Mb/s$, and the technological latency is upper bounded by $16\mu s$. Then, $R = 100Mb/s$ and $T_l = 16\mu s$. The service curve at each port of S_k is $\beta(t) = 100[t - 16]^+$.

The delay of a flow with an arrival curve α in a network element which offers a service curve β is bounded by the horizontal deviation between α and β:

$$h(\alpha, \beta) = \sup_{s \ge 0}[\inf\{\tau : \tau \ge 0 \ and \ \alpha(s) \le \beta(s + \tau)\}]$$

When there are at least two flows competing for a given output port, the maximum delay at this port is obtained by considering the overall arrival curve which is the sum of the arrival curves of all the competing flows.

A path of a PF can be modeled as the concatenation of a source node output port and several switch output ports. The Network Calculus computation starts

from the first switch along this path. Therefore the ETE delay experienced by a frame following this path is the waiting and transmission delays at the source node adding the sum of delays encountered at each output port of the crossed switches.

Let's focus on the output port of S_1 shared by pf_1 and pf_2 (Figure 1). The overall arrival curve α which is the sum of α_{pf_1} and α_{pf_2} is depicted in Figure 5. The delay $h(\alpha, \beta) = 116\ \mu s$ is computed from α and the service curve β offered by the considered output port of S_1.

3.2 Arrival Curves with Partial Synchronization of Flows

The integration of the partial synchronization of PFs in the Network Calculus approach is based on the aggregation of the flows coming from the same source node. This aggregation technique allows the integration of the *ROs* in the overall arrival curve of each switch output port.

Let's consider an output port S_k where n PFs from x source nodes compete. The computation of the overall arrival curve at this port processes as follows:

1. Classification of n PFs into x subsets based on source, where for subset SS_i, $1 \leq i \leq x$, there are n_i PFs. Then, $n_1 + n_2 + ... + n_x = n$.
2. Aggregation of the PFs of each subset SS_i as one flow, and characterization of its arrival curve α_i at S_k.
3. Computation of the overall arrival curve α at S_k which is the sum of x arrival curves $\alpha_i, 1 \leq i \leq x$ corresponding to x subsets, i.e. $\alpha = \alpha_1 + \alpha_2 + ... + \alpha_x$.

Since the Network Calculus approach propagates the computation port by port along the considered path, the proposed approach is illustrated on one example output port. The output port of S_1 where four PFs: pf_1, pf_2, pf_3 and pf_4 pass through in Figure 1 is considered.

The input PFs pf_1 and pf_2 are scheduled at the same source node N_1, and the inputs PFs pf_3 and pf_4 are scheduled at the same source node N_2. These four PFs compete for the output port of S_1. Then, $n = 4$, and there are two subsets:$SS_1 = [pf_1, pf_2]$ and $SS_2 = [pf_3, pf_4]$. So, $x = 2$, $n_1 = 2$, and $n_2 = 2$.

pf_1 and pf_2 are aggregated as one flow since they both come from N_1. Similarly, pf_3 and pf_4 are aggregated. The next step is to characterize the arrival curves α_1 and α_2 of the two aggregated flows.

The *DOs* of the four PFs are arbitrarily fixed to $O_{d,1}^{N_1} = 0\ \mu s$, $O_{d,2}^{N_1} = 1500\ \mu s$, $O_{d,3}^{N_2} = 0\ \mu s$ and $O_{d,4}^{N_2} = 500\ \mu s$. From the formula 1, the *ROs* at the source nodes are $O_{r,1,2}^{N_1} = 1500\ \mu s$, $O_{r,2,1}^{N_1} = 2500\ \mu s$, $O_{r,3,4}^{N_2} = 500\ \mu s$ and $O_{r,4,3}^{N_2} = 1500\ \mu s$.

Let's first analyze the flow aggregating pf_1 and pf_2. The arrival curves α_{pf_1} and α_{pf_2} at the output port of S_1 have been determined in Section 3.1.

In order to generate the arrival curve of the aggregated flow of pf_1 and pf_2 at S_1, we first take pf_1 as the bPF and compute the *RO* $O_{r,1,2}^{S_1} = 1470\mu s$ (the computation is detailed in Section 2). The integration of this *RO* shifts the arrival curve of pf_2 to $\alpha'_{pf_2}(t) = \alpha_{pf_2}(t - O_{r,1,2}^{S_1})$. It is depicted in Figure 6(a).

Then one *Possible Arrival Curve (PAC)* of the aggregated flow of pf_1 and pf_2 is obtained by summing α_{pf_1} and $\alpha^{'}_{pf_2}$ as depicted in Figure 6(a). It is denoted $\alpha_1^{pf_1,\{pf_2\}}$ since pf_1 is the bPF.

Obviously, for the same aggregated flow, pf_2 can be regarded as the bPF, and there is another *PAC* $\alpha_1^{pf_2,\{pf_1\}}$ of this aggregated flow at this output port. This second *PAC* is depicted in Figure 6(b).

In a general case, m flows have to be aggregated, leading to m *PACs*. Define a set $\mathcal{J} = \{1, 2, ..., m\}$, then $\alpha_i^{pf_x,\{pf_y\}}$ $(x \in \mathcal{J}, y \in \mathcal{J} \setminus x)$ represents one *Possible Arrival Curve (PAC)* of the aggregated flow of subset SS_i when pf_x is the bPF.

For pf_3 and pf_4, the *ROs* at S_1 are computed: $O^{S_1}_{r,3,4} = 500\ \mu s$, $O^{S_1}_{r,4,3} = 1440\ \mu s$. Their aggregated flow has also two *PACs* $\alpha_2^{pf_3,\{pf_4\}}$ and $\alpha_2^{pf_3,\{pf_4\}}$.

Therefore, there are four combinations of arrival curves, and each combination gives a possible overall arrival curve, which is the sum of $\alpha_1^{pf_m,\{pf_n\}}$ and $\alpha_2^{pf_x,\{pf_y\}}$ and denoted $\alpha_{m,x}$, at this port as shown in Figure 7. The overall arrival curve α is the highest among these possible overall arrival curves. In the case of switch S_1, $\alpha(t) = max(\alpha_{1,3}(t), \alpha_{2,3}(t), \alpha_{2,4}(t), \alpha_{1,4}(t))$.

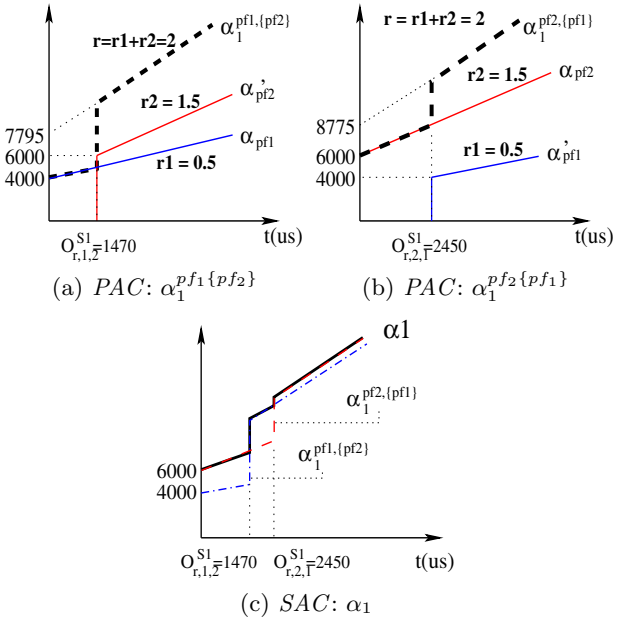
In the general case, at one output port of a switch S_k, the total number of combinations for the overall arrival curve is $N = n_1 \times n_2 \times ... \times n_x$. If each $n_i \geq 10$ and $x \geq 10$, then $N \geq 10^{10}$, which is a huge number of computation and can become intractable. In order to solve this problem, we propose an over-approximation of the arrival curve of the aggregated flow at the output port of the switch based on the following definition.

Definition 3. *At any output port, m serialized PFs from the same source node could be classified to one subset SS_i, and generates m Possible Arrival Curves, denoted $\alpha_i^{pf_x,\{pf_y\}}$ with $\mathcal{J} = \{1, 2, ..., m\}$, $x \in \mathcal{J}, y \in \mathcal{J} \setminus x$, when considering each serialized PF as the benchmark PF. Safe Arrival Curve (SAC) α_i of this subset is piecewise max of the m Possible Arrival curves, i.e. $\alpha_i(t) = max\{\alpha_i^{pf_x,\{pf_y\}}(t)\}$, for $t \geq 0$.*

By definition, the *Safe Arrival Curve* is worse than any *Possible Arrival Curve* because it always takes the largest value at any time among all the possibilities. Considering the specific situation of pf_1 and pf_2 described above, $m = n_1 = 2$, $\alpha_1(t) = max\{\alpha_1^{pf_1,\{pf_2\}}(t), \alpha_1^{pf_2,\{pf_1\}}(t)\}$. We obtain the *SAC* α_1 of the subset SS_1 shown in Figure 6(c).

The *SAC* for pf_3 and pf_4 is obtained by a similar process. By summing these two *Safe Arrival Curves*, we obtain the overall arrival curve at this output port of S_1 in Figure 8(a).

Figure 8(b) compares the arrival curve obtained from the *Safe Arrival Curves* with the four combinations of arrival curve presented in Figure 7. α in Figure 8(b) represents the overall arrival curve at $S1$ when considering the influence of the partial synchronization, while $\alpha^{'}$ is the overall arrival curve without considering that. The upper bounded delays at this port are given by $h = 156\ \mu s$ for α and $h^{'} = 236\ \mu s$ for $\alpha^{'}$. Obviously, $h < h^{'}$, so the proposed approach should provide tighter upper bound of ETE delay.



(a) PAC: $\alpha_1^{pf_1\{pf_2\}}$ (b) PAC: $\alpha_1^{pf_2\{pf_1\}}$

(c) SAC: α_1

Fig. 6. The *Possible Arrival Curves* and the *Safe Arrival Curve* of the aggregated flow

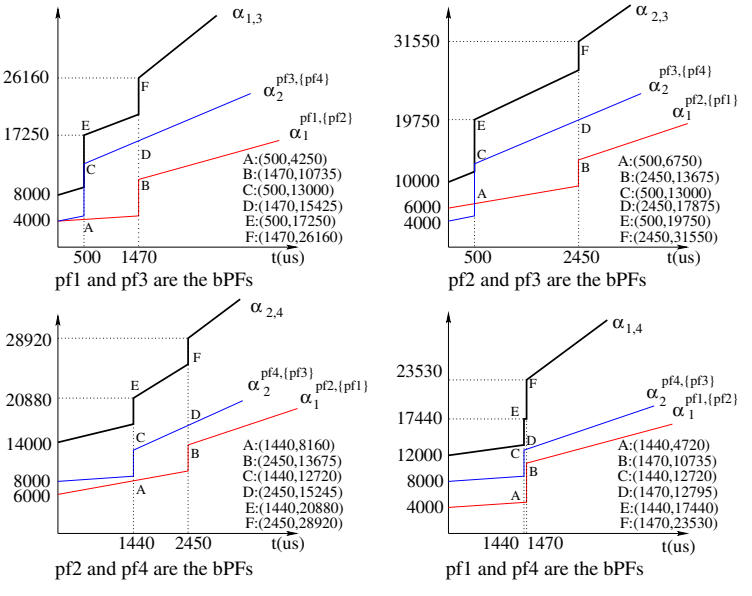
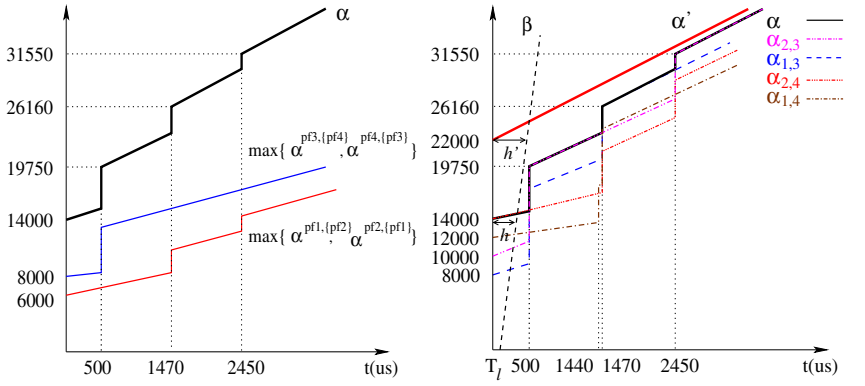


Fig. 7. Four combinations of the arrival curves



(a) The overall arrival curve α (b) Comparison

Fig. 8. The overall arrival curve and the comparison with other possibilities

The next section evaluates the improvement on ETE delay provided by the integration of the partial synchronization in the Network Calculus approach on some classical switched Ethernet architectures.

4 Evaluation of the Proposed Approach

The evaluation needs an offset assignment algorithm which is used by each source node. The algorithm considered in this paper is presented in the next paragraph.

4.1 Example of an Offset Assignment

Different algorithms have been proposed (see for instance [8], [9]). Defining such an algorithm is out of the scope of this paper. We simply choose for the switched Ethernet network an algorithm which has been proposed for automotive CAN network in [10].

This algorithm considers a discrete time system based on a time interval granularity g. The release time of each frame and the period of each PF are multiple of g. Then time intervals can be allocated to the frames of the different PFs transmitted by the source node.

For n PFs from the same source node, the assignment of offsets is over discrete time interval $[0,\frac{P_{max}}{g})$, where $P_{max} = max_{1\leq k\leq n}\{P_k\}$, and the offset for each PF pf_k is chosen in the interval $[0,P_k)$. The sequence of frame transmissions is stored in an ordered array R, which has $\frac{P_{max}}{g}$ elements. The n PFs are numbered by increasing value of their periods and proceeded from pf_1 to pf_n. The first PF pf_1 is assigned with the $DO\ O^{N_i}_{d,1} = 0$. The offset assignment of pf_k $(k > 1)$ is achieved by the following steps:

1. look for one of the longest least loaded interval in $[0, P_k)$. The first and last possible release time of the interval are noted by B_k and E_k respectively.

Table 2. The example of offset assignment

Time	0	2	4	6	8	10	12	14
$PRT\ i$	1	2	3	4	5	6	7	8
$R[i]$	$f_{1,1}$	$f_{2,1}$	$f_{1,2}$	$f_{3,1}$	$f_{1,3}$	$f_{2,2}$	$f_{1,4}$	$f_{4,1}$

PRT_i is the abbreviation of *Possible release time* i
$R[i]$ records the released frames

2. then set the offset $O_{d,k}^{N_i}$ in the middle of the selected interval, and the corresponding possible release time is r_k.
3. finally store the frames of pf_k in the array R:

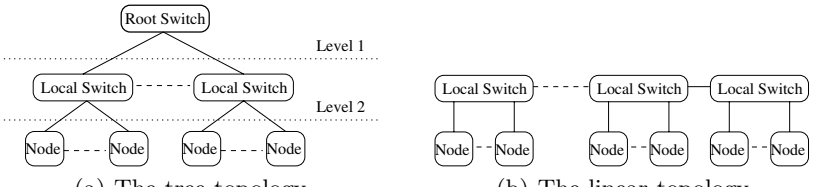
$$\forall i \in \mathbb{N}\ and\ r_k + i \cdot \frac{P_k}{g} \leq \frac{P_{max}}{g},\ do\ R[r_k + i \cdot \frac{P_k}{g}] = R[r_k + i \cdot \frac{P_k}{g}] \cup f_{k,i+1}$$

This algorithm is illustrated by the following small example. Four PFs pf_1, pf_2, pf_3, pf_4 scheduled at N_i with periods of $4ms$, $8ms$, $16ms$, $16ms$ respectively are sorted by increasing periods and $P_{max} = 16ms$. pf_1 is assigned with offset $O_{d,1}^{N_i} = 0$ which means $R[1] = f_{1,1}$. According to step 3, $R[3] = f_{1,2}$, $R[5] = f_{1,3}$ and $R[7] = f_{1,4}$. Then for pf_2, $B_2 = 2$ and $E_2 = 3$ (step 1), thus $r_2 = 2$. The array R is update with $R[2] = f_{2,1}$ and $R[6] = f_{2,2}$ (step 3). The same process is done for pf_3 and pf_4, the R array is listed in Table 2, from which we can see that $O_{d,2}^{N_i} = 2\ ms$, $O_{d,3}^{N_i} = 6\ ms$ and $O_{d,4}^{N_i} = 14\ ms$.

The algorithm presented in this section calculates the *Definitive Offsets* associated to PFs.

4.2 Obtained Results

The tree topology and linear topology are mainly used to build switched Ethernet architectures [11,12,13]. With a tree topology, nodes are connected to a local switch which is connected to a upper-layer switch or a root switch (Figure 9(a)). When there is only one switch, it is the star topology. The disadvantage of this topology is that it demands maximum cabling which is of concern to automation technology. The linear topology consists of a cascade of switches which are linked two by two (Figure 9(b)). This topology reduces the costs of infrastructure, but because of higher load per link, it generates an unfavorable topology for real-time systems.



(a) The tree topology (b) The linear topology

Fig. 9. Topologies of switched Ethernet

Table 3. Tree topology

Configuration	switch	node	PF	layer	maxL
config.1	1	10	100	1	2
config.2	4	30	300	2	4
config.3	13	90	900	3	6

Table 4. Linear topology

Configuration	switch	node	PF	maxL
config.1	1	10	100	2
config.2	2	20	200	3
config.3	3	30	300	4
config.4	4	40	400	5
config.5	5	50	500	6

The evaluated configurations of these two topologies are listed in Table 3 and Table 4. The columns *switch*, *node* and *PF* respectively mean the total number of switches, nodes and PFs in this configuration. The column *layer* means the total number of layers in the tree topology. The column *maxL* means the maximum path length. The total number of PFs increases with the total number of nodes which increases with the total number of switches.

The evaluation for each configuration is repeated, and each time the path and parameters of each PF are chosen randomly. The period of each PF is randomly chosen from the range $[1\ ms, 128\ ms]$; The L_{max} and L_{min} of each PF are randomly chosen from the ranges $[500Bytes, 1500Bytes]$ and $[100Bytes, 500Bytes]$ respectively. The transmission rate of link is $R = 100\ Mb/s$ and the technological latency of each switch is upper bounded by $T_l = 16\ \mu s$.

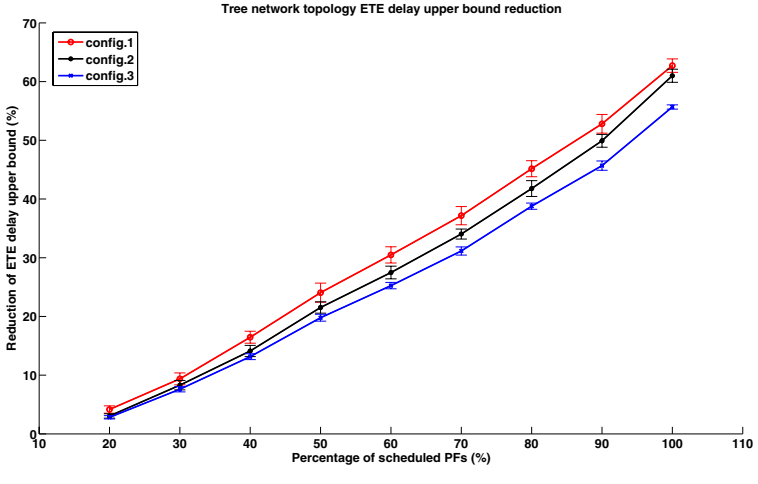


Fig. 10. Reduction of ETE delay upper bounds with tree topology

The offset assignment presented in Section 4.1 is applied to a subset of the PFs at each node of each configuration. The upper bounds of ETE delay are computed using the modified Network Calculus approach. Figure 10 shows the average obtained results with the tree topology, and Figure 11 shows those with the linear topology. The X axis represents the ratio of scheduled PFs to the total number of PFs, which describes the extend of partial synchronization. The Y

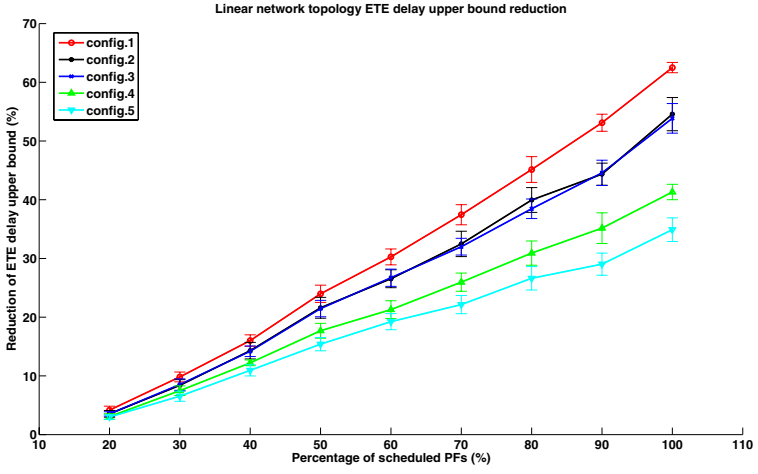


Fig. 11. Reduction of ETE delay upper bounds with linear topology

axis gives the reduction of ETE delay upper bound with partial synchronization compared to the case without synchronization.

The obtained curve shows that the partial synchronization of PFs clearly reduces the computed upper bounds for each configuration. This reduction increases with the percentage of scheduled PFs. Among different configurations of a given topology, the reduction reduces with the increase of the total number of switches and nodes. It means that the influence of partial synchronization is more obvious on simple configurations than that on complex ones. Moreover, the offsets brought by this partial synchronization work better on tree topology than on linear topology within the scope of evaluated configurations.

The mean value and standard deviation of each obtained ETE upper bound reduction are listed in Table 5 and Table 6. They are also marked besides each point in Figure 10 and Figure 11. These small standard deviations imply that the improvements of end-to-end delay upper bounds are constant and stable.

Table 5. Mean and standard values of the obtained results with tree topology

Network Config.	Param.	Percentage of scheduled PFs (%)								
		20	30	40	50	60	70	80	90	100
Config.1	mean (%)	4.15	9.36	16.46	24.05	30.48	37.17	45.16	52.80	62.71
	st.dev (%)	±0.62	± 1.01	±1.02	±1.62	±1.55	±1.36	±0.78	±1.59	±1.14
Config.2	mean (%)	3.05	8.29	14.11	21.52	27.48	34.03	41.78	49.91	60.9
	st.dev (%)	±0.37	±0.79	±0.95	±0.96	± 1.07	±0.85	±1.34	±1.08	±1.11
Config.3	mean (%)	2.82	7.60	13.15	19.78	25.25	31.14	38.78	45.68	55.67
	st.dev (%)	± 0.27	± 0.44	±0.50	±0.58	±0.54	±0.69	± 0.53	±0.79	±0.36

Table 6. Mean and standard values of the obtained results with linear topology

Network Config.	Param.	Percentage of scheduled PFs (%)								
		20	30	40	50	60	70	80	90	100
Config.1	mean (%)	4.16	9.82	16.01	23.97	30.26	37.44	45.14	53.10	62.50
	st.dev (%)	±0.67	± 0.83	±0.98	±1.46	±1.35	±1.70	±2.19	±1.45	±1.14
Config.2	mean (%)	3.53	8.39	14.30	21.59	26.54	32.47	39.95	44.35	54.58
	st.dev (%)	±0.57	±0.96	±1.40	±1.77	± 1.52	±2.16	±2.11	±1.87	±2.82
Config.3	mean (%)	3.52	8.52	14.19	21.46	26.71	32.00	38.46	44.57	53.86
	st.dev (%)	± 0.36	± 0.97	±0.89	±1.39	±1.46	±1.42	± 1.66	±2.15	±2.52
Config.4	mean (%)	3.06	7.48	12.22	17.68	21.28	25.96	30.91	35.16	41.31
	st.dev (%)	± 0.49	± 0.81	±0.48	±1.26	±1.51	±1.55	± 2.06	±2.61	±1.30
Config.5	mean (%)	2.99	6.54	10.94	15.40	19.23	22.14	26.62	29.02	34.89
	st.dev (%)	± 0.38	± 0.87	±0.95	±1.12	±1.37	±1.54	± 1.99	±1.89	±2.01

5 Conclusion and Future Work

In this paper, the partial synchronization of periodic flows brought by the local clock of each source node is presented. In order to evaluate the influence of this partial synchronization on worst-case delay analysis in the context of switched Ethernet, an approach is developed by introducing the scheduling of periodic flows (PFs) in the Network Calculus approach. The scheduling is modeled by the offset associated to each PF. With a basic offset assignment algorithm, the analysis of classical examples of switched Ethernet configurations shows that the computed ETE delay upper bounds are clearly reduced.

Further studies should be conducted on more general and complex switched Ethernet configurations so as to confirm the improvement brought by partial synchronization of flows on ETE delays done by the Network Calculus approach. The exact worst-case ETE delay can be computed by model checking for a small scale network in [14]. It could help to evaluate the pessimism introduced by the proposed approach. Moreover, further studies also deal with the introduction of partial synchronization of flows in other approaches for worst-case ETE delay analysis, such as the trajectory approach [4].

As switched Ethernet is used in avionic embedded system (AFDX: Avionics Full DupleX Switched network), evaluation of partial synchronization of flows in an AFDX context is underway.

References

1. Cruz, R.: A Calculus for Network Delay, part 1, part 2. Trans. Inf. Theory. 37, 114–141 (1991)
2. Le Boudec, J.-Y.: Application of Network Calculus To Guaranteed Service Networks. Trans. Inf. Theory. 44, 1087–1096 (1998)

3. Le Boudec, J.-Y., Patrick, T.: Network Calculus: A Theory of Deterministic Queuing Systems for the Internet. Springer, Heidelberg (2001)
4. Martin, S., Minet, P.: Schedulability Analysis of Flows Scheduled With FIFO: Application To the Expedited Forwarding Class. In: IPDPS 2006, Rhodes Island, pp. 8–15 (2006)
5. Cholvi, V., Echague, J., Le Boudec, J.-Y.: On the Feasible Scenarios at the Output of a FIFO Server. IEEE Communication Letters 9 (2005)
6. Loeser, J., Haertig, H.: Low-Latency Hard Real-Time Communication over Switched Ethernet. In: ECRTS 2004, Catania, Italy, pp. 13–22 (2004)
7. Charara, H., Scharbarg, J.-L., Ermont, J., Fraboul, C.: Method for Bounding End-to-End Delays on an AFDX Network. In: ECRTS 2006, Dresden, Germany, pp. 192–202 (2006)
8. Goossens, J.: Scheduling of Offset Free Systems. J. Real-Time Systems 24, 239–258 (2003)
9. Grenier, M., Goossens, J., Navet, N.: Near-optimal Fixed Priority Preemptive Scheduling of Offset Free Systems. In: RTNS 2006, Poitiers, France (2006)
10. Grenier, M., Havet, L., Navet, N.: Pushing the Limits of CAN - Scheduling Frames with Offsets Provides a Major Performance Boost. In: 4th European Congress on Embedded Real Time Software, Toulouse, France (2008)
11. Jasperneite, J., Neumann, P., Theis, M., Watson, K.: Deterministic Real-Time Communication with Switched Ethernet. In: WFCS 2002, Vsters, Sweden, pp. 11–18 (2002)
12. Georges, J.P., Rondeau, E., Divoux, T.: Evaluation of Switched Ethernet in an Industrial Context by Using Network Calculus. In: WFCS 2002, Vsters, Sweden, pp. 19–26 (2002)
13. Fan, X., Jonsson, M., Jonsson, J.: Guaranteed Real-Time Communication in Packet-Switched Networks with FCFS Queuing. J. The International Journal of Computer and Telecommunications Networking 53, 400–417 (2009)
14. Adnan, M., Scharbarg, J.-L., Ermont, J., Fraboul, C.: Model for Worst Case Delay Analysis of an AFDX Network using Timed Automata. In: WiP, ETFA 2010, Bilbao, Spain (2010)

Analyzing End-to-End Functional Delays on an IMA Platform

Michaël Lauer[1], Jérôme Ermont[1], Claire Pagetti[1,2], and Frédéric Boniol[2]

[1] Université de Toulouse - IRIT - INPT/ENSEEIHT
[2] Université de Toulouse - ONERA

Abstract. The Integrated Modular Avionics (IMA) platform is the latest generation of embedded architecture, in which functions share both the execution and communication resources. Functions execute in predefined time slots and communicate through an AFDX network. The purpose of the analysis presented is the verification of freshness requirements on the data exchanged between IMA applications.

The two contributions of this paper are : (1) a modeling approach for IMA platforms based on networks of timed automata. For small models, it is possible to compute exact evaluation of temporal properties using symbolic reachability analysis, (2) the collaborative use of efficient methods for worst case traversal time (WCTT) computation on the AFDX network, which results are injected in the timed automata model to help the functional analysis.

1 Introduction

1.1 Context

The concept of Integrated Modular Avionics (IMA) architecture has been proposed in the early 90s and is embedded in particular in the A380 and B787 families. The main idea is to share the computation and the communication resources in order to reduce the weight and the maintenance. The shared calculators, called *modules*, have dedicated operating systems normalized by the standard Arinc 653 [ARI97]. Practically, Arinc 653 segregates avionics functions executing on a same module, both physically and temporally. Physically, each function owns its proper memory zones. Temporally, each function executes on distinct pre-defined time slots. The communication between modules is realized via a network compliant to the standard Arinc 664[1] [ARI02]. Arinc 664, based on Ethernet technology, segregates the data flows produced by different functions.

Thus, an IMA platform can be seen as a set of modules, switches and links. A *loaded IMA platform* is an IMA platform on which a set of functions is allocated. In avionics, the configuration is static, meaning that the allocation of the functions on the resources is fully defined off-line. To fulfill the high level

[1] In A380, the implementation is the AFDX, for Avionics Full Duplex Switched Ethernet. In the sequel, we will often refer to the AFDX.

T. Margaria and B. Steffen (Eds.): ISoLA 2010, Part I, LNCS 6415, pp. 243–257, 2010.

requirements, the loaded platform must satisfy several properties, and the validation of these properties must response to the certification authorities criteria. This includes some mathematical demonstrations of real-time properties. Generally, the real-time properties are decomposed in : (1) the study of the functional behavior of a module, this includes the analyze of the schedulability and the adequacy with the pre-defined slots, (2) the evaluation of the network traversal time, that is the time for a frame to cross the network from source to destination and (3) the combination of the two analyzes: knowing the worst cast traversal time, it is possible to validate the coherence between data used in some treatment and more generally the freshness of data. The objective of the paper is to define a formal modeling of a loaded IMA platform and a general methodology for studying the functional delays[2].

1.2 Objective: Evaluation of Functional Delays

Usually, the papers that refer to *end-to-end delays*, such as [CSEF06, FS06, MM06] for instance, focus on the delays on the network, that is the time elapsed from the output of a module to the reception by an other module. *Functional delays* compute the time from the functional production to the functional consumption. If the data crosses a network, the functional delay includes the network traversal time and possibly several if the data crosses several modules.

For computing these delays, we first need to formalize the loaded IMA platform and determine the data paths. The traceability of the path followed by a data is necessary to compute functional delays and freshness. The authors of [SB07] study functional data paths on IMA platform. They allocate the functions on the resources in order to respect requirements of safety. However, they do not consider the real-time aspects and do not evaluate delays.

In [CB06], the authors analyze functional delays in a distributed system linked by a switched Ethernet network. They model the system as a network of timed automata and several abstractions are proposed to tackle the problem of combinatorial explosion. However, there are two shortcomings for our objective : (1) the architecture is not based on IMA hypothesis (2) the abstractions remain insufficient for realistic system. Thus, to the best of our knowledge, there is no existing comprehensive IMA platform formalization including temporal behavior and there is no global method for computing functional delays.

1.3 Contribution

To illustrate our purpose, we use all along the paper an understandable example of a loaded IMA platform (section 2). The platform is composed of four modules and two switches. Five functions cohabit on the platform and can be seen as

[2] This work is supported by the French National Research Agency within the SATRIMMAP (Safety and time critical middleware for future modular avionics platforms) project : http://www.irit.fr/satrimmap/

a simplified Navigation and Guidance System. For this example, we want to analyze the freshness of some data.

Formal Modeling. The first contribution is to provide a formal modeling of a loaded IMA platform. This is done using a network of timed automata. The modeling approach is modular, since two different IMA platforms, with different physical and functional architectures, can be represented by assembling basic modeling elements provided in a library. The model can be simulated using the tool UPPAAL. For small models, it is also possible to compute exact evaluation of temporal properties using symbolic reachability analysis (section 4.1). Unfortunately, as models grow bigger, we run into combinatorial explosion problems. This is the reason why we provide a solution to the scalability issue by mixing model-checking technique with trajectory approach.

Methodology for evaluating functional delays. Most of the complexity of the model comes from the network. In practice, the Arinc 664 works as follows: the frames produced by the functions are sent (resp. received) in some virtual link (VL). A VL corresponds to a predefined multicast path through the switched network, where each output port is a queue. This behavior is complex and is full of delays. Despite the use of the VL, the network traversal times are non deterministic.

Several techniques of worst case traversal time (WCTT) evaluation have been successfully applied on Arinc 664. The complexity is good enough for real configurations but at the price of an over-approximation. Network calculus [LBT01, FFG09] is used within the AFDX network and the results are shown to the certification authorities. The authors of [BD10] have proposed an extension of the network calculus to take into account the scheduling of functions on a module, but the scheduling was of type rate monotonic which does not fit with the IMA pre-defined slots. The *real-time calculus* [TCN00] and the *trajectory approach* [MM06] can be used to compute WCTT. Collaborative methods between real-time calculus and timed automata have been proposed in [LPT09]. Model-checking is used to derive flows characteristics from timed automata behavior. The usability of this approach on large systems is an open question.

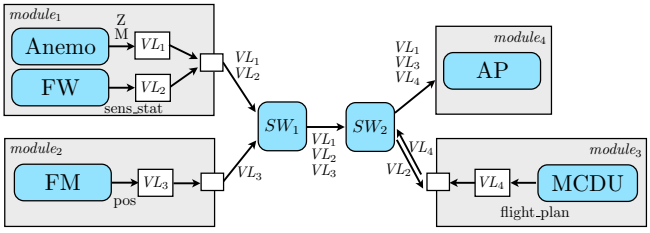
The previous methods evaluate efficiently WCTT while we want to compute functional delays. We propose to mix efficient methods for WCTT computation on the AFDX network and model-checking. For this, we extract information for the WCTT computation and we inject the obtained results in the timed automata model to help the functional analysis. We build an abstraction of the network : each *VL* is modeled with a timed channel which defines a communication link between a sender and a receiver. This channel imposes a delay in $[BCTT, WCTT]$ to the communication, where BCTT stands for best cast traversal time. Thus all behaviors of the concrete model are included in the abstract model. This methodology can be applied with any technique of WCTT evaluation and in the paper, we have chosen the trajectory approach [MM06] (section 4.2). At last, we propose an extension of the trajectory approach to leverage some pessimism in order to obtain tighter WCTT evaluation (section 4.3). The point is to avoid scenarios that cannot happen.

2 A Simplified Navigation and Guidance System

Let us consider a simplified version of a Navigation and Guidance System which aims at controlling and guiding an aircraft. It is composed of five functions: (1) the *Autopilot* function (AP) elaborates flight commands to reach an attitude defined by the next way-point of the flight plan. It requires regular updates on the position, altitude and speed of the aircraft, (2) the *Multifunction Control Display Unit* (MCDU) is the interface between the system and the crew. It allows the crew to define or to modify the flight plan, and it informs the crew of system failures, (3) the *Flight Management* function (FM) manages the flight plan, and periodically sends to AP the next way-point (*pos*) to reach, (4) the *Flight Warning* function (FW) reports on equipment status (*sens_stat*) to MCDU and finally (5) the *Anemometer* (Anemo) computes and broadcasts speed (M) and altitude (Z) of the aircraft to AP. These five functions are mapped onto an IMA platform composed of 4 core processing modules and 2 switches (cf. figure 1(a)).

Each module is managed by a real-time operating system compliant to the Arinc 653 standard [ARI97]. According to this standard, each function executes periodically within a partition. A partition is group of time slices in a Major Frame (MAF) on a module. Moreover, for the sake of determinism, executions are often composed of three consecutive steps: (1) a reception step (Rx) where the function acquires its input data, (2) a processing step (Proc) where it computes its new output and internal data, and (3) a transmission step (Tx) where it emits its output data to the module output. These three steps execute sequentially in each period. We consider in this article that each step is executed within a fixed time interval with respect to the beginning of the module MAF. The timing characteristics of the five partitions above are defined in figure 1(b). For instance, the time interval $[12, 14]$ associated with the Rx step of partition FW means that FW acquires its input at least 12 and at most 14 millisecond after the beginning of the MAF of module 1. In the same way, FW emits its output data between 17 and 19. The periodic execution of ANEMO and FW is depicted on figure 1(c).

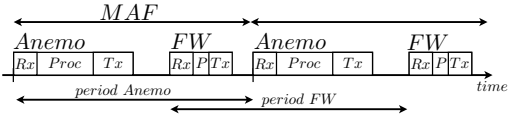
According to the IMA principles, partitions exchange data through an AFDX protocol [ARI02]. As explained in introduction, this protocol is based on Virtual Links (VL). Data transmitted through VL are encapsulated into frames. The allocation of data in frames and on VL is statically defined off-line. A VL defines a static route between a source partition and one or several destination partitions. In order to limit traffic bursts in the network, each VL input port implements a leaky bucket traffic shaper defined by its BAG (Bandwidth Allocation Gap) parameter: the minimum time interval between two successive frames emitted by the input port of the VL is equal to BAG time units. For the sake of simplicity, we consider that VL output ports (in each destination partition) are sampling ports: whenever a frame arrives at a destination module, the data transmitted by the frame are extracted and are stored in the input buffer of the destination partition, the previous data (from the previous frame) being replaced by these new data. Figure 1(a) presents the VL definition of the case study. For instance, Z and M are transmitted by VL_1 from *Anemo* to AP via switches SW_1 and



(a) Allocation of the functions

Func	Per.	Rx	Proc	Tx
Anemo	20	[0,3]	[3, 8]	[8,10]
FW	20	[12,14]	[14,17]	[17,19]
FM	20	[0,5]	[5,15]	[15,19]
MCDU	20	[0,5]	[5,17]	[17,20]
AP	15	[0,4]	[4,12]	[12,15]

(b) Function characteristics



(c) Static scheduling of $module_1$

Fig. 1. Loaded IMA platform

SW_2. We suppose however that Z and M are emitted in two distinct frames. We also assume that (1) all *VLs* have the same BAG value $b = 2ms$, (2) all frames have the same size $s = 4000$ bits, and (3) all physical links work at 100Mbits/s.

Given these specifications, the property we want to study is the following: the worst case age of Z (altitude) consumed by AP is less than $25ms$.

3 Formal modeling

3.1 The Model

An IMA system is composed (1) an hardware architecture, (2) a functional architecture, and (3) a mapping of the latter on the former. Let us note M, S and L the sets of modules, switches, and links composing the hardware architecture. Let us note Pa, D, Fr and V the sets of partitions, data, frames and virtual links composing the functional architecture.

Modules and switches. A module or a switch $x \in M \cup S$ has ports. Let us note $x.input.i$ (resp. $x.output.j$) the i^{th} (resp. j^{th}) input (resp. output) port of x. The mapping specification of partitions on modules is denoted by a $part(m)$ which associates each module $m \in M$ with a set of partitions in Pa. $p \in part(m)$ means that p is executed on m.

Links. A link connects an output port of a module or a switch to an input port of another module or switch. It is characterized by a physical bandwidth. We denote $conn(l) =< x.output.i, y.input.j >$ to mean that $l \in L$ connects the i^{th} output port of $x \in M \cup S$ the to the j^{th} input port of $y \in M \cup S$. We also note $bw(l)$ the bandwidth (in Mbits/s) of link l.

Partitions, data, and frames. Partitions produce and consume data. Data are encapsulated in frames, and frames are transmitted through virtuals links. Let us note $in(p)$ and $out(p)$ the set of data consumed and produced by $p \in Pa$. We denote by $data(fr)$ the set of data encapsulated in the frame $fr \in Fr$, *frame(vl)* the set of frames transmitted on the virtual links $vl \in V$ and $vl\_part(p)$ the set of VLs of p.

Virtual links. A virtual link is defined by both its static path through the nodes of the architecture, and its BAG. We define $path(vl) = [n_1, \ldots, n_k]$ a function which associates each virtual link $vl \in V$ with its physical path. This physical path is an ordered sequence of nodes $n_i \in M \cup S$ (modules or switches) interconnected by links from L. We define then $BAG(vl)$ the BAG value of vl.

Scheduling. We define the scheduling of $p \in Pa$ by a period and two time intervals (for the two steps Rx and Tx). The scheduling specification of p is $sched(p) =< \pi, [Rx\_min, Rx\_sup], [Tx\_min, Tx\_sup] >$ where π is the period of p and the other parameters are the intervals Rx and Tx. The *sched* function of the case study is given figure 1(b).

Given the sets M, S, L, Pa, D, Fr and V, a system $syst$ is completely defined by the ten functions :

$$syst =< part, conn, bw, in, out, data, frame, path, BAG, sched >$$

We define the behavior of such a system by expressing it as a network of extended timed automata [AD94, LPY97]. We associate a functional and temporal behavior with each element.

3.2 Behavioral Description with Timed Automata

Modules behavior. A module contains a set of partitions periodically scheduled within a time frame. The global behavior of a module results then in the composition of the behavior of each partition, combined with the behavior of the communication layer (data encapsulation and frames transmission through VL traffic shaper). From a formal point of view, this global behavior is defined as a network of timed automata presented (in case of $module_1$) figure 2, and composed of:

- a scheduler automaton which periodically activates the partitions;
- an automaton for each partition;
- an automaton for modeling the encapsulation of each data into each frame. Frame transmissions are represented by a signal $frame[i][j]$ where i identifies a network component and j identifies the frame. For instance, $frame[1][0]$ represents the arrival of frame 0 in the component 1 (VL_1 traffic shaper) and $frame[6][0]$ the transmission of this frame to the component 6 ($module_1$ output port);
- an automaton modeling the behavior (i.e. the traffic shaper) of each VL;
- an automaton modeling the behavior of each output port.

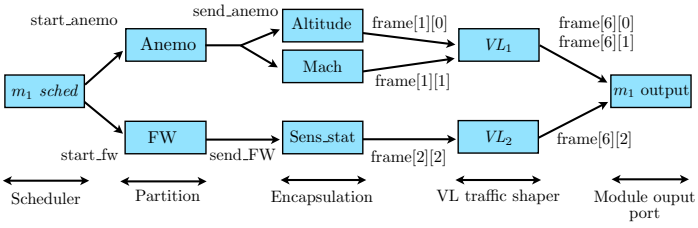


Fig. 2. *Module*$_1$ modeled by a network of timed automata

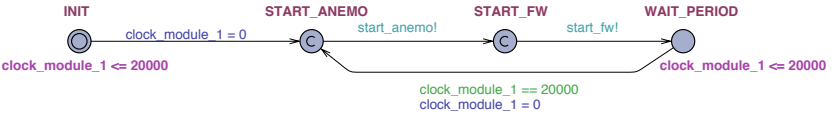
The scheduler automaton. According to the Arinc 653 standard, partitions are scheduled on each module within a periodic MAF (Major Frame). This MAF, implemented by the *sched* function, associates to each partition a period and a set of time intervals defining when the partition reads and writes its inputs and its outputs. The scheduler automaton models the periodic activation of partitions allocated on each module. The timing aspects (i.e., the time intervals Rx and Tx) are then modeled in each partition automaton. Figure 3(a) models the scheduling of *Anemo* and *FW* during a periodic *MAF* of $20000\mu s$.

Partitions automata. After being activated by the scheduler, each partition p runs in its own time slices defined by $sched(p)$. Let us consider partition *Anemo*. Its behavior is defined by the timed automaton on the top of Figure 3(b) (in this example, only the data transmission is considered). The transmission occurs (node *SEND*) during [8000, 10000] μs. The system sends the altitude and the speed as depicted on the bottom of Figure 3(b).

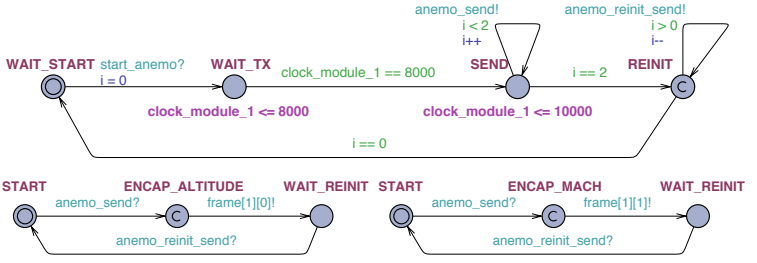
Switch/module output port automata. A switch/module output port implements a FIFO queuing policy. The transmission time of a frame depends on its size and the port throughput. A frame is transmitted to the next element on its path. For a frame i and an output port j, $SERVICE\_TIME[i][j]$ gives the service time of i and $ROUTE[i][j]$ gives the next element crossed by it. Figure 3 shows the modeling of a generic output port. *idPort* identifies the port. *list* is a local array storing identifiers of frames.

Virtual link traffic shaper automata. A virtual link traffic shaper implements a leaky bucket algorithm. The maximum rate is one frame every BAG. Modeling is similar to output port except : (1) service time is the same for each frame and it is equal to the BAG, (2) the queue is considered empty only BAG time units after the transmission of the last frame, (3) a frame is instantly transmitted (service time is null) if the queue is empty when it arrives.

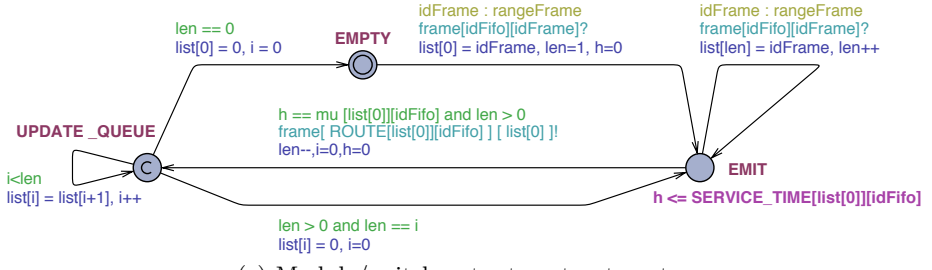
Global behavior. As expressed above, the behavior of each module can be obtained by composing several timed automata. In the same way, the global behavior of the system is obtained by composing the modules behavior together with the switches behavior. This leads to a global timed automata network modeling the exact behavior of the whole system.



(a) $module_1$ automaton



(b) Anemometer partition automaton and its encapsulation automata



(c) Module/switch output port automaton

Fig. 3. Timed automata of the model

As illustration, the timed automata network of the case study is composed of 24 automata: 4 scheduler automata (one per module), 5 partition automata (one per partition), 5 encapsulation automata (one per data), 4 VL traffic shaper automata (one per VL), 6 output port automata (3 module output ports for module 1, 2 and 3; 1 output port for switch SW_1 and 2 output ports for switch SW_2).

4 Analysis and Verification

4.1 Model-Checking Verification

The property to verify is "the age of Z must be lower than *Max_Age* when read by the autopilot". This property is expressed by a test automaton. As described in [BnA98, BBF+01], a test automaton implements an *unhappy* state which is reached if the property does not hold. Thus, the verification becomes a decidable reachability problem, which can be tackle with model-checking tools. In our model, the maximum reachable age of Z computed is $24200\mu s$. Furthermore, the model-checker gives us a scenario reaching this worst case. Let Z_1 and Z_2 be two successive values of Z. The maximum age of Z is reached when : (1)

the distance between Z_1 and Z_2 production is maximum, (2) Z_2 experiences maximum delay to reach AP and finally (3) the last reading of Z_1 coincides with Z_2 arrival in AP. Z_2 experiences maximum delay when : (1) Z_2 and a speed data (M) are produced simultaneously, (2) M is placed first in VL_1 traffic shaper, (3) Z_2 and a position data arrive simultaneously in SW_1 output port and (4) Z_2 and a flight plan data arrive simultaneously in SW_2 output port.

4.2 A Mixed Verification Technique

Model-checking is an exact computing technique which is applicable on small case studies. However this technique does not scale well. There are at least three sources of complexity : the asynchronism of the module clocks, the non-deterministic order of the frames sent by the partitions and the network queues. To reduce the complexity, we use a two-step verification methodology. First we calculate QoS properties of flows in the network with the trajectory approach, and second we abstract the network with these QoS properties.

Recalls on trajectory approach. The trajectory approach [MM06] is a technique for computing deterministic bounds on WCTT which can be applied to AFDX network [BSF09]. The network is represented as a graph where the nodes are the output ports of the modules and the switches, and the vertices are the connections between two components. The approach analyses flows of frames. A VL is a flow and is characterized by its path $path(vl)$, the minimum inter-arrival time between two successive frames $BAG(vl)$ and the maximum processing time of a frame in the nodes of its path C_{vl}. The processing time of a frame corresponds to the time required by the output of a node to emit this frame.

We briefly explain how the WCTT of VL_i can be obtained. Let $\mathcal{P}_i$ be equal to $path(VL_i) = [n_{i_1}, \ldots, n_{i_k}]$. Let f be a frame of VL_i arriving in the network at time t. The exact WCTT is : $R_i^e = \max_{t \geq 0} \left\{ LS_i^{n_{i_k}}(t) + C_i - t \right\}$, where $C_i = C_{VL_i}$ and $LS_i^{n_{i_k}}(t)$ is the latest starting time of f in its last node. Since the result of the trajectory approach does not depend on the choice of f, the WCTT of f is also the WCTT of VL_i (consequently, expressions relative to f are indexed by i). The trajectory approach bounds $LS_i^{n_{i_k}}(t)$ with:

$$LS_i^{n_{i_k}}(t) \leq \sum_{j \in X(i)} WL_j^{flow}(\delta_{i,j}(t)) + \sum_{h \in \mathcal{P}_i \setminus \{n_{i_k}\}} \left(\max_{j \in vl\_node(h)} \{C_j\} \right) - C_i \qquad (1)$$

- $X(i)$: the set of VLs crossing the path $\mathcal{P}_i$ of VL_i. Thus, $X(i) = \{j | \mathcal{P}_j \cap \mathcal{P}_i \neq \emptyset\}$;
- $\delta_{i,j}(t)$: the time interval during which the frames of j interfere with f;
- $WL_j^{flow}(\mathbf{x})$: the workload produced by j during time interval $\mathbf{x}$;
- $vl\_node(h)$: the set of VLs crossing node h;
- $\sum_{h \in \mathcal{P}_i \setminus \{n_{i_k}\}} \left(\max_{j \in vl\_node(h)} \{C_j\} \right)$: the sum of the maximum processing times for each node along the path of VL_i (except the last node);

In [MM06], $WL_j^{flow}(\mathbf{x})$ is bounded by : $\left(1 + \left\lfloor \frac{L(\mathbf{x})}{BAG_j} \right\rfloor\right) \cdot C_j$, with $L(\mathbf{x})$ the length of $\mathbf{x}$. And it is proved that $\forall (t, j) \in \mathbb{R}^+ \times X(i),\ \delta_{i,j}(t) \subseteq \Delta_{i,j}(t)$ where :

- $\Delta_{i,j}(t) = [M_i^{first_{i,j}} - S_{max_j}^{first_{i,j}}\ ,\ t + S_{max_i}^{first_{i,j}} - S_{min}^{first_{i,j}}]$;
- $first_{i,j}$: the first node where VL_i and VL_j cross each other;
- $S_{max_i}^h$: maximal time for a frame of VL_i to go from its source to h;
- $S_{min_j}^h$: minimal time for a frame of VL_j to go from its source to h;
- $sub(\mathcal{P}_i, h)$: set of all nodes that precede node h along $\mathcal{P}_i$;
- $M_i^h = \sum_{h' \in sub(\mathcal{P}_i,h)} \left(\min_{j \in vl\_node(h')}\{C_j\}\right)$: the sum of minimum processing time of the VL for each node in $sub(\mathcal{P}_i, h)$;
- $t + A_{i,j}$ denotes the length of $\Delta_{i,j}$(t).

Thus $\forall t \in \mathbb{R}^+$, $\sum_{j \in X(i)} WL_j^{flow}(\delta_{i,j}(t)) \leq \sum_{j \in X(i)} \left(1 + \left\lfloor \frac{t + A_{i,j}}{BAG_j} \right\rfloor\right) \cdot C_j$. Hence an upper bound of the WCTT of VL_i is $R_i^T = \max_{t \geq 0} \left\{ W_i^{n_{i_k}}(t) + C_i - t \right\}$ where $W_i^{n_{i_k}}(t)$ is an upper bound of the latest starting time of f in its last node:

$$W_i^{n_{i_k}}(t) = \sum_{j \in X(i)} \left(1 + \left\lfloor \frac{t + A_{i,j}}{BAG_j} \right\rfloor\right) \cdot C_j + \sum_{h \in \mathcal{P}_i \setminus \{n_{i_k}\}} \left(\max_{j \in vl\_node(h)} \{C_j\} \right) - C_i \quad (2)$$

We apply these results to determine the WCTT of VL_1 (denoted $WCTT(VL_1)$).

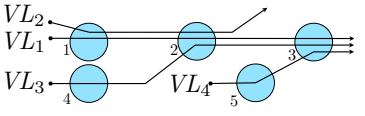


Fig. 4. Network graph

Figure 4 depicts the path $\mathcal{P}_1 = [1, 2, 3]$ and the interfering VLs $X(1) = \{1, 2, 3, 4\}$. Node 1 is $module_1$ output port, node 2 is SW_1 output port and node 3 is the output port of SW_2. Nodes 4 and 5 are the output port of $module_2$ and $module_3$ respectively.

In order to evaluate $W_1^3(t)$, we need to determine $A_{1,1}$, $A_{1,2}$, $A_{1,3}$ and $A_{1,4}$. From equation 2 we can see that $W_i^{n_{i_k}}(t)$ is a recursive expression. Indeed, $A_{i,j}$ depends on $S_{max_i}^{first_{i,j}}$ which is the WCTT of VL_i on $sub(\mathcal{P}_i, first_{i,j})$. We describe part of $A_{1,4}$ calculation. VL_1 and VL_4 first meet in node 3, thus :

$$A_{1,4} = S_{max_1}^3 - \sum_{h' \in \{1,2\}} \left(\min_{j \in vl\_node(h')} \{C_j\} \right) + S_{max_4}^3 - S_{min_4}^3 \quad (3)$$

VL_4 only crosses node 5 until node 3 and it is the only flow on this sub path, hence $S_{max_4}^3 = S_{min_4}^3 = C_4$. From equation 2 and $S_{max_1}^3$ definition we have:

$$S_{max_1}^3 = \max_{t \geq 0} \left\{ \left(1 + \left\lfloor \frac{t + A_{1,1}}{BAG_1} \right\rfloor\right) C_1 + \left(1 + \left\lfloor \frac{t + A_{1,2}}{BAG_2} \right\rfloor\right) C_2 + \left(1 + \left\lfloor \frac{t + A_{1,3}}{BAG_3} \right\rfloor\right) C_3 + \max_{j \in \{1,2\}} \{C_j\} - t \right\}$$

The first common node of VL_1 and VL_2 is their source node 1, hence $A_{1,1} = A_{1,2} = 0$. We do not detail the calculation of $A_{1,3} = C_1 = 40\mu s$. Thus, we have:

$$S_{max_1}^3 = \max_{t \geq 0} \left\{ \left(1 + \left\lfloor \frac{t}{2000} \right\rfloor\right) 40 + \left(1 + \left\lfloor \frac{t}{2000} \right\rfloor\right) 40 + \left(1 + \left\lfloor \frac{t + 40}{2000} \right\rfloor\right) 40 + 40 - t \right\}$$

Hence, $S_{max_1}^3 = 160\mu s$. Injecting these results in equation 3, we have: $A_{1,4} = 160 - (40 + 40) = 80\mu s$. We are now able to evaluate $W_1^3(t)$ and consequently $WCTT(VL_1)$. The calculation gives $WCTT(VL_1) = 240\mu s$.

Network abstraction in the timed automata model. First we project the model by only keeping elements contributing to the functional analysis. Only the partitions *Anemo* and *Autopilot* are involved in the maximum age of Z data analysis. Second, we replace the network by a timed channel whose delay is in $[BCTT(VL_1), WCTT(VL_1)] = [120, 240]$. Indeed the authors of [CB06] have shown that a switched Ethernet network which offers deterministic guarantees on traversal time can be abstracted by an end-to-end timed channel. BCTT happens when a frame of VL_1 is alone on its path : $BCTT = 3 \times C_1 = 120\mu s$.

We apply a verification technique similar to the one described in section 4.1 on the reduced system. We obtain a maximum age for Z of $24240\mu s$ which is slightly worse than the previous analysis. However, computation time is reduced dramatically : first method requires several hours while the second requires less than a second. The variation is as‡ substantial because the reduced model contains only a pair of modules.

In the worst case *sens_stat* data does not delay Z. The specific scheduling of the partitions *Anemo* and *FW* prevents their respective *VL* to interfere. Trajectory approach is not designed to take into account these specificities. The pessimism comes in particular from the ignorance of relationship between arrival times of frames (note that network calculus or real-time calculus encounter the same pessimism). In section 4.3 we introduce an extension of the trajectory approach, which allows us to take into account the specific behavior of the IMA platform. This leads to tighter WCTT evaluation.

4.3 Offset-Based Trajectory Approach

In the classical trajectory approach, all flows are assumed to be asynchronous. For two distinctive flows, there is no correlation between the arrival times of their respective frames in the network. The partitions hosted by a same IMA module share a common clock and execute sequentially according to the pre-defined slots. Thus all flows at a module output are correlated. We describe a methodology to leverage this specificity in order to obtain tighter bounds on WCTT. We rewrite inequality 1 in order to group the workload of the *VL* produced by a common module.

$$LS_i^{n_{i_k}}(t) \leq \sum_{m \in M} \sum_{j \in X(m,i)} WL_j^{flow}(\delta_{i,j}(t)) + \sum_{h \in \mathcal{P}_i \setminus \{n_{i_k}\}} \left(\max_{j \in vl\_node(h)} C_j \right) - C_i \qquad (4)$$

- $X(m,i)$: the set of *VLs* emitted by $module_m$ and crossing the path of VL_i;
- $\sum_{j \in X(m,i)} WL_j^{flow}(\delta_{i,j}(t))$ the global workload produced by $module_m$.

Our idea is to bound in inequation 4 this global workload by a maximal workload function that takes into account the scheduling of the partitions on $module_m$. With respect to classical real-time analysis this function is called a maximal interference function. For each module we define : (1) an interval of interference (a time interval where all frames delaying f are produced), (2) a maximal interference function.

Interval of Interference. As proved in [MM06], all frames of VL_j interfering with VL_i are produced during the interval $\Delta_{i,j}(t)$. We define an interval including all the $\Delta_{i,j}(t) = [a_{i,j}(t), b_{i,j}(t)]$ of the VLs of a module (we denote $t + A_i^m$ the length of this interval):

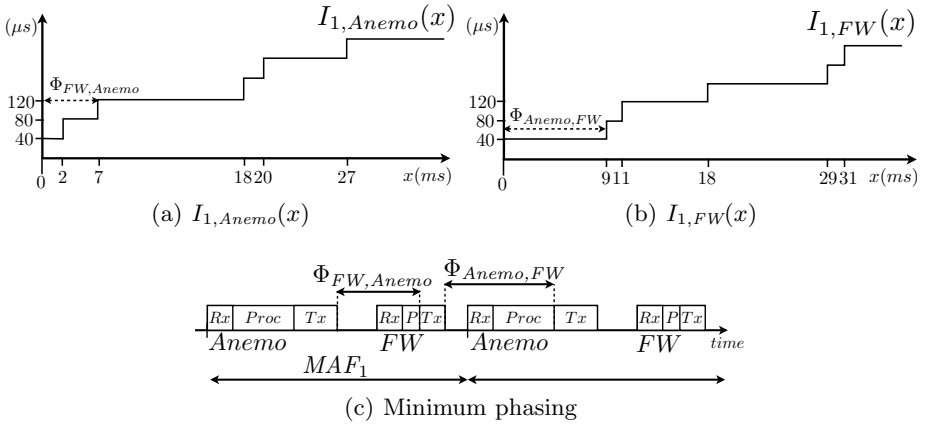
$$\Delta_i^m(t) = [\min_{j \in X(m,i)} \{a_{i,j}(t)\}, \max_{j \in X(m,i)} \{b_{i,j}(t)\}] \tag{5}$$

Hence, we have :

$$\sum_{m \in M} \sum_{j \in X(m,i)} WL_j^{flow}(\delta_{i,j}(t)) \leq \sum_{m \in M} \sum_{j \in X(m,i)} WL_j^{flow}(\Delta_i^m(t)) \tag{6}$$

Maximal Interference Function of a Module. We introduce the concept by determining the interference of $module_1$ on a frame f during x units of time. There are two cases: either a VL of $Anemo$ first interferes with f or it is a VL of FW. We call the first partition to interfere the *initiator partition*. We denote by $I_{m,c}(x)$ the interference of $module_m$ where c is the initiator partition.

Figure 5(a) and 5(b) depict $I_{1,Anemo}(x)$ and $I_{1,FW}(x)$ respectively. We use the concept of minimum phasing between partitions. It is the smallest distance between the end of the transmission step of a partition and the start of another one. Figure 5(c), $\Phi_{FW,Anemo}$ (resp. $\Phi_{Anemo,\ FW}$) is the minimum phasing between $Anemo$ and FW (resp. FW and $Anemo$).



(a) $I_{1,Anemo}(x)$ (b) $I_{1,FW}(x)$

(c) Minimum phasing

Fig. 5. Interference and minimum phasing concepts

$I_{1,Anemo}$ is constructed as follows:

- $\forall x \in [0, 2], I_{1,Anemo}(x) = C_1$. $Anemo$ sends two frames at the end of its transmission step. However VL_1 traffic shaper prevents the sending of more than one frame;
- $\forall x \in [2, 7,] I_{1,Anemo}(x) = I_{1,Anemo}(0) + C_1$. The second frame of $Anemo$ is released by VL_1 traffic shaper;

- $\forall x \in [7, 18], I_{1,Anemo}(x) = I_{1,Anemo}(2) + C_2$. Since $7ms$ is the minimum phasing $\Phi_{Anemo,FW}$, a frame can be sent by FW. Next frame release of FW occurs periodically every MAF_1;
- $\forall x \in [18, 20], I_{1,Anemo}(x) = I_{1,Anemo}(7) + C_1$. $18ms$ is the minimum distance between the first and the second transmission step of *Anemo*: it is equal to the MAF minus the length of the transmission step of *Anemo*. *Anemo* releases two frames and only one is delivered because of VL_1 traffic shaper. Next releases of frames will occur periodically every MAF_1;

We denote by $I_m(x)$ the maximal interference caused by $module_m$ (independently of the initiator partition). Thus $I_m(x) = \max_{c \in part(m)} I_{m,c}(x)$. For instance, since $\forall x \in \mathbb{R}^+$, $I_{1,Anemo}(x) \geq I_{1,FW}(x)$, we have $I_1(x) = I_{1,Anemo}(x)$.

Formalization. The formalization of the previous idea is inspired by the work on offset-based real-time analysis such as described in [MTN08]. Let us first introduce some useful notations: N_i is the maximal number of frames delivered on VL_i every MAF_m, $Tx_p = [a_p, b_p]$ is the interval of transmission of p. To compute $I_m(x)$, we must compute, for every partition c of module m, $I_{m,c}(x)$. We first introduce an intermediate function $WL_p(x)$ which returns the maximal workload produced by a partition p during x units of time.

Formula for $WL_p(x)$. $WL_p(x)$ is defined on $\mathbb{R}^+$ by:

$$WL_p(x) = \sum_{i \in vl\_part(p)} \left[\left(1 + \left\lfloor \frac{x}{MAF_m} \right\rfloor \right) \cdot N_i \cdot C_i - \left(N_i - 1 - \left\lfloor \frac{x^*}{BAG_i} \right\rfloor \right)^+ \cdot C_i \right] \quad (7)$$

- $x^* = x \mod MAF_m$;
- $\left(1 + \left\lfloor \frac{x}{MAF_m} \right\rfloor \right) \cdot N_i \cdot C_i$ is the maximum workload p can deliver to VL_i;
- $\left(N_i - 1 - \left\lfloor \frac{x^*}{BAG_i} \right\rfloor \right)^+ \cdot C_i$ regulates the previous term in order to respect VL_i traffic shaper. Its value is $(N_i - 1) \cdot C_i$ each time the partition delivers a burst of N_i frames. Then its value decreases of C_i every BAG_i, until it becomes null;
- $\forall a \in \mathbb{R}$, $(a)^+ = max\{0, a\}$.

Formula for $I_{m,c}(x)$ and $I_m(x)$. We first compute the minimum phasing between two partitions $(p, c) \in part(m)^2 : \Phi_{p,c} = (a_p - a_c + (b_c - a_c)) \mod MAF_m$ In the worst case, the initiator partition emits its frames at the end of its transmission step and the other partitions emit their frames at the start of their respective transmission step. In the worst case, a partition p starts contributing to the interference when x $= \Phi_{p,c}$. Thus we have:

$$I_{m,c}(x) = \sum_{p \in part(m)} WL_p(x - \Phi_{p,c}) \quad (8)$$

Formula for $I_m(x)$ is simply: $\forall x \in \mathbb{R}^+, I_m(x) = \max_{c \in part(m)} \{I_{m,c}(x)\}$. And we have for all time interval $\mathbf{x}$ and $module_m$: $\sum_{j \in X(m,i)} WL_j^{flow}(\mathbf{x}) \leq I_m(L(\mathbf{x}))$.

Hence, with inequalities 4 and 6, we can upper bound $LS_i^{n_{i_k}}(t)$ and $WCTT(VL_i)$ is bounded by $R_i^O = \max_{t\geq 0}\left\{WO_i^{n_{i_k}}(t) + C_i - t\right\}$ where:

$$WO_i^{n_{i_k}} = \sum_{m\in M} I_m(t + A_i^m) + \sum_{h\in\mathcal{P}_i\setminus\{n_{i_k}\}} \left(\max_{j\in vl\_part(h)} C_j\right) - C_i \tag{9}$$

Application. We apply these results to determine the WCTT of VL_1. Figure 4 depicts the path $\mathcal{P}_1 = [1, 2, 3]$ and the interfering VLs grouped by module are : $X(1,1) = \{1, 2\}$, $X(2,1) = \{3\}$, $X(3,1) = \{4\}$, $X(4,1) = \emptyset$. Since $module_4$ does not interfere with VL_1, $\forall t \in \mathbb{R}$, $I_4(t) = 0$. Thus we have :

$$WO_1^3 = I_1(t + A_1^1) + I_2(t + A_1^2) + I_3(t + A_1^3) + \sum_{h\in\{1,2\}} \left(\max_{j\in vl\_node(h)} C_j\right) - C_i \tag{10}$$

We describe part of A_1^2 calculation. From 5 we have: $A_1^2 = S^2_{max_1} - S^2_{min_3} - M_1^2 + S^2_{max_3}$. We have $S^2_{max_3} = S^2_{min_3} = C_3 = 40\mu s$, $M_1^2 = 40\mu s$ and $S^2_{max_1} = \max_{t\geq 0}\left\{I_1(t + A_1^1) - t\right\}$. The first common node of VL_1 and VL_2 is their source node 1, hence $A_1^1 = 0$. We have shown in the informal presentation that $\forall x \in \mathbb{R}$, $I_1(x) = I_1, Anemo(x)$. Hence:

$$S^2_{max_1} = \max_{t\geq 0}\left\{\sum_{p\in\{Anemo,FW\}} WL_p(t - \Phi_{p,Anemo}) - t\right\} \tag{11}$$

With $\Phi_{Anemo,Anemo} = -2000\mu s$, $\Phi_{Anemo,Anemo} = 7000\mu s$, $MAF_1 = 20000\mu s$ and $BAG_1 = BAG_2 = 2000\mu s$ we have $S^2_{max_1} = 40\mu s$. Thus $A_1^2 = 0$. We find $WCTT(VL_1) = 200\mu s$.

5 Conclusion and Perspectives

In this paper we have presented a modeling approach for IMA platforms based on networks of timed automata. On small systems, verification of complex temporal requirements and exact evaluation of temporal properties are possible. The scalability issue has been tackle with an abstraction technique. Communications in the system are abstracted with timed channels. These abstractions are built with the help on an efficient WCTT evaluation technique : the trajectory approach. However, due to specificities of the IMA platform, the WCTT is pessimistic. We proposed an extension of the trajectory approach which takes into account the Arinc 653 scheduling and thus provides tighter bounds. A tool applying our methodology is being developed.

References

[AD94] Alur, R., Dill, D.L.: Theory of Timed Automata. Theoritical Computer Science 126(2), 183–235 (1994)

[ARI97] Aeronautical Radio Inc.: ARINC 653. Avionics Application Software Standard Interface (1997)

[ARI02] Aeronautical Radio Inc.: ARINC 664. Aircraft Data Network, Part 1: Systems Concepts and Overview (2002)

[BBF+01] Bérard, B., Bidoit, M., Finkel, A., Laroussinie, F., Petit, A., Petrucci, L., Schnoebelen, P.: Systems and Software Verification. Model-Checking Techniques and Tools. Springer, Heidelberg (2001)

[BD10] Boyer, M., Doose, D.: Collaboration entre méthode d'ordonnancement et calcul réseau. Presented at the GDR GPL Day (2010)

[BnA98] Arjona, A.B.: Vérification et synthèse de systèmes temporisés par des méthodes d'obervation et d'analyse paramétrique (in english). PhD thesis, Supaero, Toulouse, France (1998)

[BSF09] Bauer, H., Scharbarg, J.-L., Fraboul, C.: Applying and optimizing trajectory approach for performance evaluation of AFDX avionics network. In: Emerging Technologies and Factory Automation, ETFA (2009)

[CB06] Carcenac, F., Boniol, F.: A formal framework for verifying distributed embedded systems based on abstraction methods. International Journal on Software Tools for Technology Transfer (STTT) 8(6) (2006)

[CSEF06] Charara, H., Scharbarg, J.-L., Ermont, J., Fraboul, C.: Methods for bounding end-to-end delays on an AFDX network. In: Euromicro Conference on Real-Time Systems, ECRTS (2006)

[FFG09] Frances, F., Fraboul, C., Grieu, J.: Using network calculus to optimize the AFDX network. In: European Congress on Embedded Real-Time Software (ERTS), Toulouse France, 25/01/06-27/01/06, SIA/3AF/SEE (2009)

[FS06] Fidler, M., Schmitt, J.B.: On the way to a distributed systems calculus: an end-to-end network calculus with data scaling. In: SIGMETRICS 2006/Performance 2006: Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems. ACM, New York (2006)

[LBT01] Le Boudec, J.-Y., Thiran, P.: Network Calculus. LNCS, vol. 2050. Springer, Heidelberg (2001)

[LPT09] Lampka, K., Perathoner, S., Thiele, L.: Analytic real-time analysis and timed automata: a hybrid method for analyzing embedded real-time systems. In: EMSOFT 2009: Proceedings of the Seventh ACM International Conference on Embedded Software. ACM, New York (2009)

[LPY97] Larsen, K.G., Pettersson, P., Yi, W.: UPPAAL in a Nutshell. International Journal on Software Tools for Technology Transfer 1(1-2), 134–152 (1997)

[MM06] Martin, S., Minet, P.: Schedulability analysis of flows scheduled with fifo: application to the expedited forwarding class. In: Guo, M., Yang, L.T., Di Martino, B., Zima, H.P., Dongarra, J., Tang, F. (eds.) ISPA 2006. LNCS, vol. 4330. Springer, Heidelberg (2006)

[MTN08] Maki-Turja, J., Nolin, M.: Efficient implementation of tight response-times for tasks with offsets. Real-Time Systems Journal (2008)

[SB07] Sagaspe, L., Bieber, P.: Constraint-based design and allocation of shared avionics resources. In: 26th AIAA-IEEE Digital Avionics Systems Conference, Dallas (2007)

[TCN00] Thiele, L., Chakraborty, S., Naedele, M.: Real-time calculus for scheduling hard real-time systems. In: ISCAS, pp. 101–104 (2000)

Tools in Scientific Workflow Composition

Joost N. Kok[1], Anna-Lena Lamprecht[2], and Mark D. Wilkinson[3]

[1] Leiden Institute of Advanced Computer Science,
Leiden University, Leiden, The Netherlands
joost@liacs.nl
[2] Chair for Programming Systems,
Technical University Dortmund, Dortmund, Germany
anna-lena.lamprecht@cs.tu-dortmund.de
[3] Heart + Lung Institute at St. Paul's Hospital,
University of British Columbia, Vancouver, BC, Canada
markw@illuminae.com

Scientific workflows are combinations of activities and computations in order to solve scientific problems. In contrast to, for instance, business workflows that implement business processes involving different persons and information systems, scientific workflows are used to carry out computational experiments, possibly confirming or invalidating scientific hypotheses [1]. Scientific workflow systems [2,3] support and automate the execution of error-prone, repetitive tasks such as data access, transformation, and analysis. Several systems for different purposes and following different approaches have been developed in the last decade, and research in this comparatively new field is currently going into many different directions.

This ISoLA 2010 special track is devoted to "Tools in Scientific Workflow Composition". Its papers comprise subjects such as tools and frameworks for workflow composition, semantically aware workflow development, and automatic workflow composition, as well as some case studies, examples, and experiences. The contributions are primarily from the bioinformatics domain, but do also contain examples from other (scientific) application domains. One group of contributions presents concrete workflow applications:

- The paper **Workflows for Metabolic Flux Analysis: Data Integration and Human Interaction** (Tolga Dalman, Peter Droste, Michael Weitzel, Wolfgang Wiechert, Katharina Nöh) describes a workflow environment for ^{13}C metabolic flux analysis (MFA) that supports the full cycle of graphical network modeling, model configuration, simulation, and visualization. This work puts a strong focus on the integration of interactive steps into the workflows, since ^{13}C MFA relies on human expert knowledge in several parts of the analysis process.
- In **Intelligent Document Routing in SQL: a Case Study** (Carlos Soares, Miguel Calejo) the authors describe the development of a document-routing workflow for a large institution. They apply data-mining techniques to establish correlations between document-content and previous routings, and then use this knowledge for the routing of new documents. This paper

T. Margaria and B. Steffen (Eds.): ISoLA 2010, Part I, LNCS 6415, pp. 258–260, 2010.

describes well the difficulties of a small data mining project and includes the results of a simple implementation.
- The paper **Combining Subgroup Discovery and Permutation Testing to Reduce Redundancy** (Jeroen S. de Bruin, Joost Kok) investigates the benefits of scientific workflows for a genomics experiment that requires intensive computing and parallelization. The results show that simple workflow parallelization can lead to substantial optimizations in rule redundancy elimination.

Another group of papers within this special track addresses the challenge of semantically supported workflow design and management. The Semantic Web [4] aims at thoroughly equipping individual data and services with machine-processable meta-information in order to simplify the discovery of relevant resources. The usefulness of properly semantically annotated data and services has been recognized by the life science community earlier than by other application domains, and thus various projects have made significant progress towards a Semantic Web for bioinformatics [5,6]. Three papers in the track apply semantic information about types and services for supporting workflow development:

- In **Workflow composition and enactment using jORCA** (Johan Karlsson, Victoria Martín-Requena, Javier Ríos, Oswaldo Trelles) the authors present the latest improvements to the jORCA system for semantically-guided workflow construction. Making use of the Magallanes framework, jORCA is able to automatically or semi-automatically construct complete workflows given an input and desired output data type. Moreover it is possible to store and execute the created pipelines of web services. In the discussion section the authors make some important observations regarding the applicability of automatic service composition techniques and the structure and size of required meta-data ontologies.
- The paper **Workflow Construction for Service-Oriented Knowledge Discovery** (Vid Podpečan, Monika Žakova, Nada Lavrač) describes a framework for semi-automatic creation of data mining workflows. An existing data-mining tool suite is made available in a service-oriented fashion using web services technology, and combined with a knowledge discovery ontology, and planning and ontological reasoning techniques.
- A comparison between two Taverna plug-ins for semantically-guided service discovery is given in the paper **Semantically-guided Workflow Construction in Taverna: The SADI and BioMoby Plug-ins** (David Withers, Edward Kawas, Luke McCarthy, Benjamin Vandervalk, Mark Wilkinson). The plugins embed a functionality to find suitable successors for workflow nodes into the workflow system, chaining tools based on output-input-type matching.

Finally, the track contains a contribution that is not directly concerned with semantically supported workflow development, but rather with the documentation and publication of in-silico experiments:

– **A Linked Data Approach to Sharing Workflows and Workflow Results** (Marco Roos, Sean Bechhofer, Jun Zhao, Paolo Missier, David Newman, Dave de Roure, M. Scott Marshall) identifies the problem of providing a digital equivalent of the "Materials and Methods" section (as custom in biological wet lab publications) for bioinformatics publications. The proposed linked-data approach to tackle this problem is based on the idea to formally describe bioinformatics experiments using workflows, web-based information repositories, and Semantic Web technology. The authors describe a prototype end-to-end linked data model for discovering and annotating workflows, and for capturing and representing provenance information when those workflows are executed.

We think that the seven papers in this track give a good impression about current topics in research on scientific workflows. In closing, we would like to thank all contributors and the conference organizers for their efforts, which made this special track on "Tools in Scientific Workflow Composition" possible. We hope that you will enjoy the presentations, the discussions with your colleagues, and your visit to Crete.

References

1. Ludäscher, B., Weske, M., McPhillips, T., Bowers, S.: Scientific Workflows: Business as Usual? In: Dayal, U., Eder, J., Koehler, J., Reijers, H.A. (eds.) Business Process Management. LNCS, vol. 5701, pp. 31–47. Springer, Heidelberg (2009)
2. Ghanem, M., Curcin, V.: Scientific workflow systems - can one size fit all? In: Cairo International Biomedical Engineering Conference, CIBEC 2008, pp. 1–9 (2008)
3. Zhao, Y., Raicu, I., Foster, I.: Scientific Workflow Systems for 21st Century, New Bottle or New Wine? In: Proceedings of the 2008 IEEE Congress on Services - Part I, pp. 467–471. IEEE Computer Society, Los Alamitos (2008)
4. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web - A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities. Scientific American 284(5), 34–43 (2001)
5. Cannata, N., Schroder, M., Marangoni, R., Romano, P.: A Semantic Web for bioinformatics: goals, tools, systems, applications. BMC Bioinformatics 9(suppl. 4), S1 (2008)
6. Burger, A., Paschke, A., Romano, P., Splendiani, A.: Semantic Web Applications and Tools for Life Sciences 2008. In: Proc. of 1st Workshop SWAT4LS 2008. CEUR Workshop Proceedings, Edinburgh, United Kingdom (November 2008)

Workflows for Metabolic Flux Analysis: Data Integration and Human Interaction

Tolga Dalman, Peter Droste, Michael Weitzel, Wolfgang Wiechert, and Katharina Nöh

Institute of Biotechnology 2, Forschungszentrum Jülich, 52425 Jülich, Germany
{t.dalman,p.droste,m.weitzel,w.wiechert,k.noeh}@fz-juelich.de

Abstract. Software frameworks implementing scientific workflow applications have become ubiquitous in many research fields. The most beneficial advantages of workflow-enabled applications involve automation of routine operations and distributed computing on heterogeneous systems. Particular challenges in scientific applications include grid-scale orchestration of complex tasks with interactive workflows and data management allowing for integration of heterogeneous data sets.

We present a workflow for the ^{13}C isotope-based Metabolic Flux Analysis (13C-MFA). The core of any 13C-MFA study is the *metabolic network modeling* workflow. It consists of sub-tasks involving model set-up and acquisition of measurement data sets within a graphical environment, the evaluation of the model equations and, finally, the visualization of data and simulation results. Human intervention and the integration of various knowledge and data sources is crucial in each step of the modeling workflow. A scientific workflow framework is presented that serves for organization and automation of complex analysis processes involved in 13C-MFA applications. By encapsulating technical details and avoiding recurrent issues, sources for errors are minimized, the evaluation procedure for ^{13}C labeling experiments is accelerated and, moreover, becomes documentable.

Keywords: Scientific Workflows, Human Tasks, Database Integration, 13C-MFA, SOA.

1 Introduction

In recent years, scientific workflows emerged as a key technology in a growing number of research fields. The most beneficial advantages of workflow-enabled scientific applications involve automation of routine operations and distributed computing on heterogeneous systems. Scientific workflow systems have been realized successfully in various research fields [13,7,11,14] and nowadays a plenitude of successful workflow applications are available [16,12,1]. The diversity of these examples show: scientific workflow frameworks are inherently domain-dependent.

Current challenges in the workflow field include Grid-scale orchestration of complex, long-running tasks with interactive workflows. We present an application in the field of Metabolic Engineering where all of these challenges need

T. Margaria and B. Steffen (Eds.): ISoLA 2010, Part I, LNCS 6415, pp. 261–275, 2010.

to be addressed. In particular, we show that domain-specific requirements for a scientific workflow framework managing Metabolic Engineering applications is of central importance.

1.1 Metabolic Flux Analysis with Labeling Experiments

The major objective in Systems Biology and Metabolic Engineering is to understand complex processes in biological organisms [10,18]. In these research fields a combination of huge amounts of diverse data, advanced analytical tools, and the use of mathematical models aims at a targeted *in silico* design of optimized cell factories.

The major functional determinants of cell physiology are the *in vivo* reaction rates, i.e. the velocity in which certain substances are transformed into another. Thus, the understanding of the microorganisms' metabolic pathways is the key for a directed cell engineering towards the next generation of, e.g., high-performance producing strains. The intracellular reaction rates, however, cannot be directly observed. Instead, the rates, or synonymously fluxes, have to be indirectly estimated by a model-based analysis of stable isotope patterns from carbon labeling experiments, called ^{13}C-Metabolic Flux Analysis (13C-MFA) [21,26,28].

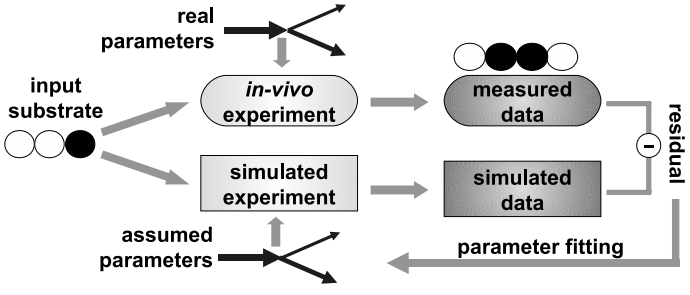


Fig. 1. Principle of Metabolic Flux Analysis with carbon labeling experiments

Current practice of 13C-MFA involves a variety of steps: post-processing of measured raw data from labeling experiments, computer-aided knowledge acquisition of metabolic pathways, the employment of sophisticated software tools for modeling and specially-tailored visualization for the interpretation of analysis results. A typical workflow can be sketched as follows:

1. In a carbon labeling experiment, living cells are fed with specifically ^{13}C-labeled substrates. The distribution of labeling patterns within intracellular metabolites is detected with highly sensitive mass spectrometry devices. Subsequently, these measurements are post-processed and collected in spread sheets.

2. Organism-specific information about metabolic reactions is acquired from publicly available resources and expert knowledge. Based on this information, a metabolic model is set up.
3. Mass balance equations are automatically generated that describe the metabolites' labeling patterns in detail. Various *in silico* methods are utilized to explore the capabilities of the model for different parameter settings.
4. By comparison of the experimentally acquired measurement data with the simulated measurement data, the real flux distribution is estimated from an iterative parameter estimation approach.

The 13C-MFA procedure shown in Fig. 1 is reviewed in more detail elsewhere [26,28]. In real world applications, this workflow is far from being a straightforward procedure, but involves many iterative cycles. Thus, a scientific workflow system organizing 13C-MFA processes in an automated fashion is needed.

1.2 Scientific Workflow Applications in the 13C-MFA Domain

The term *scientific workflow* emphasizes the extended needs of workflow applications in the research domain [22]. Contemporary topics closely related to Grid computing and scientific workflows include:

- *Workflow orchestration*: a scientific workflow engine is responsible for the invocation and distribution of tasks onto (compute) resources. In the scientific community, Taverna [7] and Kepler [13] are widely known workflow engines.
- *Grid provisioning*: on-demand acquisition of Grid resources is a desirable feature of a scientific workflow framework. Various applications address the integration of resource provisioning into workflow frameworks [2,14,24].
- *Data management*: scientific applications often require the handling of large amounts of data. The integration of scientific data into workflow and Grid frameworks has been presented in various projects [19,1].

Each scientific domain imposes specific requirements onto a scientific workflow framework. These non-standard requirements for workflow-enabled 13C-MFA applications have been identified:

(a) *Human interaction*: 13C-MFA workflows typically involve manual tasks consciously performed by human experts. The scientific workflow framework needs to be capable of modeling and integrating human actor steps in an appropriate way.
(b) *Dynamic workflows*: because typical 13C-MFA workflows consist of several steps interconnected by conditional branches and loops, a *control-based* workflow modeling scheme is preferred to solutions with an emphasis on *data-driven* workflow modeling [13,20]. The capability of modeling fault handlers (e.g. automatic retry of failed tasks) is of special interest for large-scale scientific workflows [17,9].
(c) *Domain-specific graphical user interfaces*: assistance with the modeling procedure as well as the visualization of simulation results is crucial for user acceptance and success. Specially tailored techniques emerged in the field of 13C-MFA (cf. Fig. 6 for an example) [5].

The BPEL-based open-source solution *ActiveBPEL* was elected as *control-based* scientific workflow engine, allowing fine-grained modeling of large-scale 13C-MFA applications. Several valuable *ActiveBPEL* extensions are readily available, like legacy code wrapping, fault-tolerant workflow execution and on-demand Grid integration [8,9,2]. Because BPEL-defined workflows are exposed as web services in a service-oriented architecture (SOA), 13C-MFA applications can be composed of reusable sub-workflows.

1.3 Aims of this Contribution

A scientific workflow framework supports users of scientific applications by hiding technical aspects, and thus significantly reduces the complexity of applications. Even more important, processing steps become reproducible and, in turn documentable. This contribution focuses on two important aspects in the context of 13C-MFA workflows: data integration and graphical user interfaces. In Section 2, the modeling and visualization software *Omix*, the simulation toolbox *13CFLUX2*, and the scientific workflow architecture are introduced. Section 3 presents the modeling workflow as application example in detail. For the integration of existing 13C-MFA software tools into the scientific workflow framework, two aspects of implementation are focused on in detail in Section 4: first, a plug-in interface for *Omix* is presented; second, the web service integration of *13CFLUX2* into the workflow framework is demonstrated by means of the parameter fitting procedure. Finally, Section 5 concludes this contribution.

2 Ingredients for 13C-MFA

Because the 13C-MFA method is a model-based approach, adequate software tools supporting the modeling process are required. In the following, a scientific workflow framework for 13C-MFA applications is presented. This workflow system utilizes the graphical model editing and visualization software *Omix*, and the high-performance simulation toolbox *13CFLUX2*.

2.1 Model Editing and Visualization with Omix

In order to set up models of biochemical organisms in an appropriate way, expert knowledge is necessary. Therefore, modelers make heavy use of various knowledge sources, such as published literature and reaction databases. Beside the modeling process of metabolic networks, the second important task of 13C-MFA is the visualization of intermediate and final results from experimental and simulation studies. *Omix* is a highly sophisticated metabolic network editor which is developed for modeling and visualization purposes [4]. In *Omix*, results from experiment and simulation are displayed in the context of the same graphical network representation. The software is developed in Java using the Jambi wrapper of the Qt framework for the graphical user interface.

2.2 High-Performance Simulation Toolbox: 13CFLUX2

13CFLUX2 is a set of high-performance software programs implementing the 13C-MFA method [25]. *13CFLUX2* consists of about 20 distinct applications written in C++, Perl and Python. These programs are command-line applications, which can be categorized as follows:

- *Tools for Simulation and Exploration*: programs for simulation form the core of the *13CFLUX2* toolbox. Other tools in this category include Monte Carlo sampling methods for the characterization of the solution space, flexible parameterization of system equations and various methods for advanced stoichiometric analysis.
- *Tools for Parameter Fitting*: based on a model and a set of measurements, flux parameters are estimated. Different optimization strategies with specific parameterizations can be selected.
- *Tools for Statistical Analysis*: linearized statistics tools are implemented here [27], but also nonlinear Monte Carlo methods are available for estimating confidence regions of the determined flux parameters. Tools for optimal experimental design are also included [15].
- *Miscellaneous Tools*: various tools for data management, consistency checking and conversion are also part of *13CFLUX2*.

2.3 Scientific Workflows for 13C-MFA

The scientific workflow framework presented here is a recent development that aims at the adoption of scientific workflow automation to 13C-MFA applications. This framework integrates *13CFLUX2*, *Omix* and expert knowledge into scientific workflow applications.

Many software packages consist of a plenitude of programs, database and user interfaces usually developed independently. This often results in a heterogeneous software environment, making automation efforts with workflows difficult to implement. Program interfaces provided by *13CFLUX2* and *Omix* are designed for web service extensibility, thus allowing optimal integration into a distributed environment (cf. Section 4.2). Having set up a 13C-MFA SOA environment, web service-enabled programs can be orchestrated by a scientific workflow framework.

An overview of the 13C-MFA workflow architecture is depicted in Fig. 2. A database middleware provides web service access to measurement and reaction databases (top). Using this middleware interface, 13C-MFA workflows (i.e. modeling, simulation and visualization workflows) can also access public databases, e.g. KEGG. The simulation workflows (i.e. exploration, parameter fitting and statistics) are exposed as web and Grid services. With *Omix*, models for 13C-MFA simulations are graphically edited and results are visualized (middle, left to right). Scientific workflows are orchestrated using *ActiveBPEL* (bottom).

3 Metabolic Reaction Network Modeling Workflow

Having the main ingredients for 13C-MFA introduced, the core modeling workflow is selected to demonstrate workflow realization within our scientific workflow

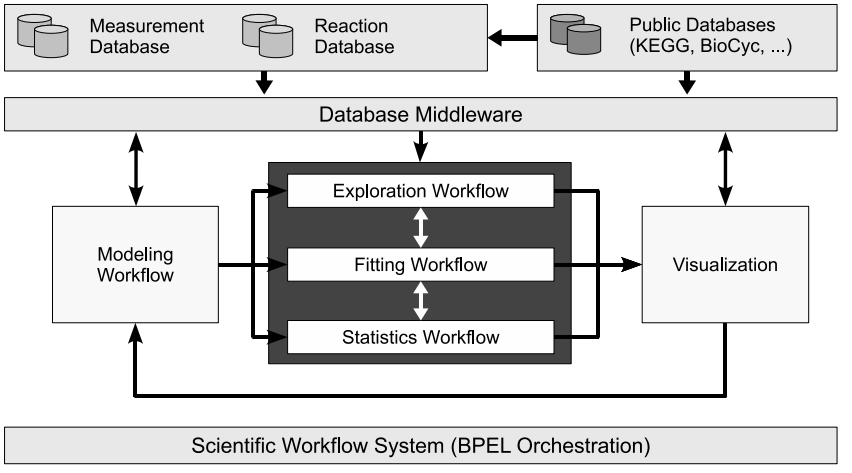


Fig. 2. The general architecture of the scientific workflow system for 13C-MFA. The building blocks are the scientific workflow system (bottom), applications (middle) and database middleware (top).

system. The modeling process basically consists of the following four steps (cf. Fig. 3):

1. **Graphical Modeling:** biochemical networks are either derived from existing models or built from scratch. This step not only handles the setup of the network topology, but also the specification of the carbon atoms' fates. Biological knowledge and various information sources are incorporated into the model, such as published data from literature or public databases.
2. **Model Configuration:** the network structure is extended by different classes of stoichiometric equality and inequality constraints. One class simply restricts the directionality of selected biochemical reactions, other classes tie together certain reaction rates or limit the range of allowed flux values.
3. **Simulation and Evaluation:** assuming that the stoichiometric constraints defined in the previous step are feasible, simulated measurements are obtained from the solution of the model equations. Combining the simulation results with measurement data obtained from isotope labeling experiments, the unknown fluxes are then estimated using an iterative, computationally intensive fitting approach. Subsequently, confidence regions of the estimated fluxes are derived using statistical methods.
4. **Visualization:** finally, simulation results are visualized. Therefore, *Omix* is employed as a specialized toolkit allowing data visualization in association with metabolic network diagrams.

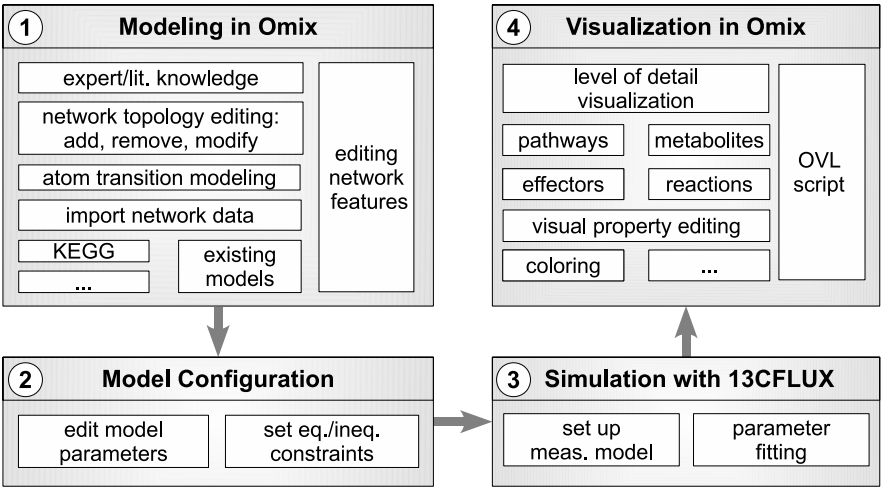


Fig. 3. A typical 13C-MFA modeling workflow. This workflow consists of the four steps model editing, configuration, parameter estimation with measurement data and visualization of simulation results.

The four steps of this workflow are discussed in detail in the following sections.

3.1 Graphical Network Modeling with Omix

An elegant way to prevent technically involved and, thus, error-prone manual editing of model description documents is the graphical editing of metabolic network models. These models are constituted by enzyme-catalyzed biochemical reactions converting substrate pools to intermediate and product pools. Key properties for the quantitative understanding of cellular metabolism are absolute metabolite *concentrations* and the reaction rates, the *fluxes*. Both, concentrations and fluxes are correlated quantities. In a graphical modeling process the modeler inserts pool and reaction symbols into a diagram and interconnects them with lines (cf. Fig. 4). The modeled network structure is easy to grasp because of its visual representation. Hence, errors in the model can be discovered very fast.

Omix mimics the functionality found in other popular vector drawing tools (see Fig. 5 a)). This warrants intuitive access to the software and eases the modeling of metabolic networks, even for users having little experience with computational tools. In principle, there are two ways to set up a network model in *Omix*:

(a) The network diagram can be drawn manually bottom up, i.e. every pool and reaction symbol is drawn individually and the connection lines are inserted one by one. Typical sources of information are network diagram taken from scientific publications or expert knowledge.
(b) Network topologies can be imported from already existing files or reaction databases and merged in whole or in part into the edited document.

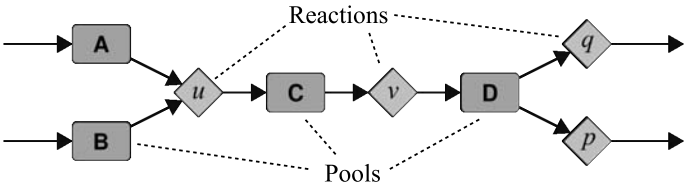


Fig. 4. Scheme of a metabolic network model consisting of metabolite pools (**A**,**B**,**C**,**D**) converted by biochemical reactions (u,v,p,q)

Both methods can be combined in any order to built up models of metabolic networks. To accelerate the drawing process, *Omix* offers several semi-automatic graph drawing techniques.

An important feature of *Omix* is that the software is equipped with a plug-in interface. The utilization of the plug-in interface facilitates the integration of *Omix* into the scientific workflow framework. For example, an *Omix* plug-in for database connectivity to KEGG[1] enables the user to inspect the metabolic networks provided by the KEGG database, to gather and select network parts for importing them into the *Omix* document. The *Omix* plug-in interface is described in more detail in Section 4.1.

3.2 Network Model Configuration

An XML-based document format called *FluxML* was developed for applications in the *13CFLUX2* environment [25]. Beside network topology information, FluxML documents contain full information required for model parameterization including stoichiometric constraints, atom transitions and measurement specifications.

In order to generate a FluxML document from the drawn network model, an *Omix* plug-in manages the import/export of FluxML documents including a comprehensive validation functionality. Additionally, a network graph is augmented with other essential information:

- *atom transitions*: carbon atom mappings of each reaction are defined in the document.
- *measurement specification*: labeling patterns for input substrates are specified. In addition, measurement values can be stored.
- *constraint and parameter configurations*: parameters constraining the network specification and flux parameter values are supplied.

The *Omix* FluxML plug-in offers dialog windows for editing all of these parameters. E.g. atom transitions can be edited graphically using the FluxML plug-in (cf. Fig. 5 b). Alternatively, parameters are adopted from an imported document. Inconsistent parameters or an invalid network topology are rejected by the validation tool included in the FluxML plug-in.

[1] http://www.genome.jp/kegg

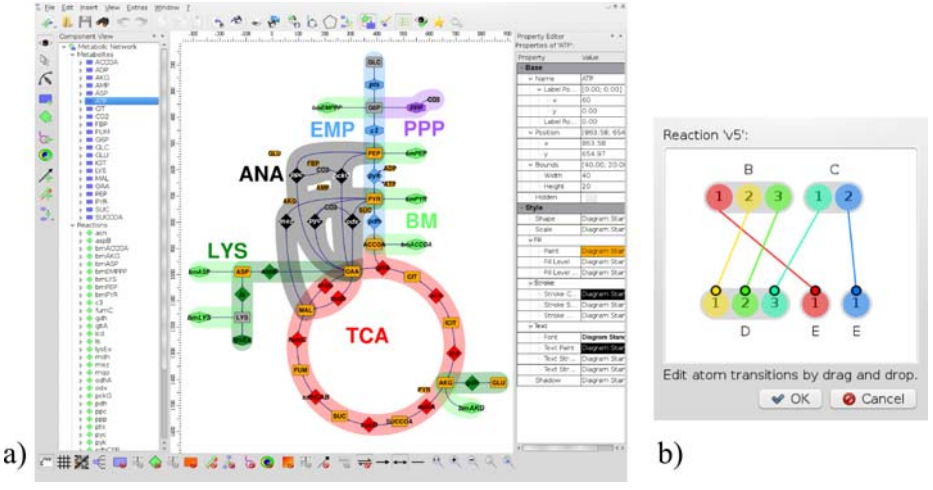


Fig. 5. Screenshot of *Omix* — a) The central window component is the drawing area for network diagrams. Toolbars containing icons around this component make various editing options available. Sidebars provide information about network properties. b) Atom transitions of the reactions can be edited graphically with *drag and drop*.

3.3 Simulation and Evaluation

Given a valid metabolic network model, fluxes can be estimated with *13CFLUX2*. For successful execution of the parameter fitting process, measurements from isotope labeling experiments need to be incorporated into the FluxML document. Simulation and parameter estimation of high-dimensional and nonlinear 13C-MFA models are computationally sophisticated procedures (cf. Fig. 1) described elsewhere [25]. Subsequent to the parameter fitting, a statistical analysis is conducted to identify the determined flux distribution's certainty. In particular, confidence intervals of all fluxes are determined.

3.4 Visualization

Simulation results are finally visualized in the *Omix* network diagram (cf. Fig. 6 for typical flux distribution maps). Visualization in *Omix* is programmable by a scripting language called *Omix Visualization Language* (OVL) and therefore fully customizable to any user requirement. In particular, OVL allows a fast and simple access to visual properties of network symbols like color, shape and line width, just to mention a few. For instance, OVL allows the user to implement interactive components, to parse data files and to assign data to visual properties of the network diagram ([4,3]).

Fig. 6 a) shows a network diagram representing the central metabolism of a microbial organism. The diagram contains metabolite pools and fluxes of central metabolic pathways connected by arrow-headed lines. If reactions are reversible,

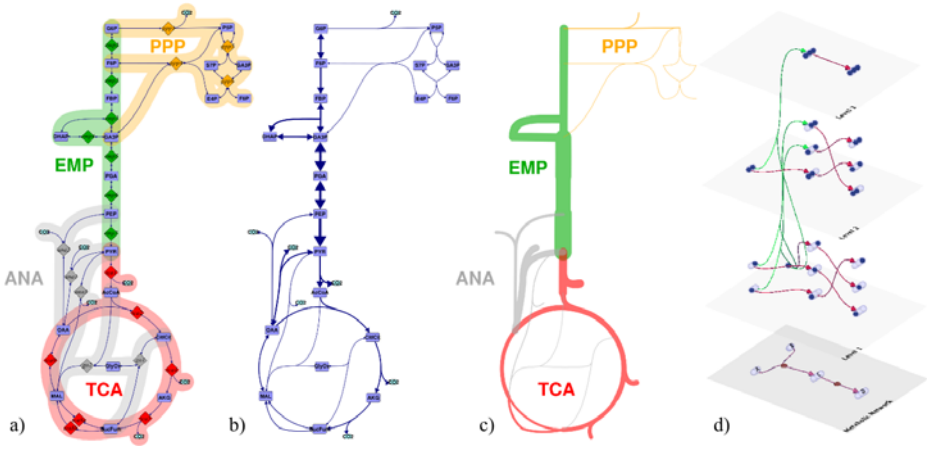


Fig. 6. Visualization in the context of MFA – a) metabolic network diagram with highlighted pathways (e.g. gluconeogenesis - green, pentose phosphate pathway - orange, citric acid cycle - red etc.); boxes show metabolites; diamond symbols represent reactions where their color codes for the pathway the reaction is assigned to. b) and c) show the network in two different levels of detail augmented with flux distribution data from a 13C-MFA. d) shows the atom transitions of a simple example network visualized in 3D by the *Omix* plug-in CumoVis.

i.e. without preferred directionality, this is indicated visually by a double arrow-headed line. For ease of recognition, biochemical reactions are often grouped to metabolic pathways. This is indicated by highlighting color strokes in the background of the pathway.

Because network diagrams can be large and very complex, *Omix* offers the option to reduce the level of detail of a network diagram by hiding whole classes of network components. In the visualization example, flux symbols as well as pathways are hidden (Fig. 6 b) and all details except pathways are concealed (Fig. 6 c), respectively.

Beside showing different levels of detail, the network is augmented with data from the simulation study. In the example, the flux value is mapped to the width of the connection lines of a reaction: line width indicates the velocity of the conversion rate (cf. Fig. 6 b). The width of the pathway stroke indicates the overall activity of reactions in the pathway (cf. Fig. 6 c). A global view of the carbon flow over the whole network is facilitated utilizing another *Omix* plug-in called CumoVis [5] (cf. Fig. 6 d).

4 Implementation Details

Having presented the workflow components for the modeling of a metabolic network, we now focus on two specific integration aspects of *Omix* and *13CFLUX2* into the scientific workflow framework.

4.1 Omix Plug-In Interface

Omix provides an interface which allows to extend the facilities of the software. This interface is called the *Omix API*. An *Omix* plug-in is a Java archive containing Java plug-in code and a specification file. The specification file describes the plug-in interface in detail, including plug-in name, license information, version and type of the plug-in. The following extension types can be realized as Omix plug-ins:

- **Interaction Extension** equip the *Omix* main window with graphical and interactive components, e.g. with menus or toolbars as document-based or document independent feature.
- **Model I/O File Filters** handle file import and export.
- **Image Export Filters** are used to implement the export of graphics into various image and animation formats, like SWF.
- **Network Communication Protocols** implement Internet transfer interfaces. For example, the network protocols SSH or SMB are available as web communication plug-ins.
- **Data Type Management** for editing plug-in-specific data types, i.e. model parameters (cf. Section 3.1).
- **Plug-in Configuration** attach custom configuration window components to the main configuration manager window of *Omix*.

An *Omix* plug-in can realize and combine an arbitrary number of these extension types. The FluxML plug-in, for instance, realizes an interaction extension and hereby provides the network validation feature in the menu bar. Simultaneously, it realizes a network I/O filter and a data type manager for the parameter modeling feature. The *KEGG Database Import* likewise implements an interaction extension using the privilege to establish database connectivity and to compose new documents in *Omix*. Because this plug-in interface is well documented any third party can develop plug-ins in order to extend Omix with specific functionality.

4.2 Web Service Implementation of the Parameter Fitting Program

As a typical *13CFLUX2* program, the parameter estimation program *fitfluxes* is chosen to exemplarily demonstrate web service extensibility. This Unix console application takes several program arguments. For basic functionality, the following arguments are required to be specified:

-i [file]: input FluxML model
-o [file]: output XML file with estimated flux parameters

Several optional command-line arguments are available, e.g. for tuning the optimization procedure:

-O [string]: select optimizer package
-g [string]: select gradient mode

A typical call of *fitfluxes* is:

```
fitfluxes -i model.fml -o out.xml -O IPOPT -g analytic
```

Here, the input (*model.fml*) and output (*out.xml*) files are specified, the optimizer package *IPOPT* [23] and the *analytic gradient* mode are used. Converting this program into a web service is accomplished by wrapping the call by a Java web service program:

```
@WebService()
public class FitFluxesWS {
  @WebMethod(operationName = "fitfluxes")
  public int fitfluxes(
    @WebParam(name = "infile") String fml,
    @WebParam(name = "outfile") String fwd,
    @WebParam(name = "optimizer") String opt,
    @WebParam(name = "gradient") String grad )
  {
    String fitfluxesCmd = ...; /* assemble the command */
    /* run fitfluxes */
    prc = Runtime.getRuntime().exec(fitfluxesCmd);
    /* handle failures, process results */
    return 0;
  }
}
```

Listing 1.1. Java web service definition for *fitfluxes*

Being an illustrative example for wrapping existing command-line programs with web services, the general procedure imposes several limitations:

- The web service is assumed to share a common networking file system. However, in a real world Grid environment, this is typically not the case. Thus, data exchange has to be modeled as well.
- A web service wrapper for each *13CFLUX2* program needs to be implemented. This manual procedure is an inflexible, laborious and, hence, error-prone procedure.
- Unix I/O streams need to be modeled as well. For example, *13CFLUX2* programs write error logging messages to the *standard error stream*.

Several solutions for wrapping legacy application code into a SOA are available, e.g. GEMLCA, Soaplab or GMS [22]. The *Legacy Code Description Language* (LCDL) is elected, because *13CFLUX2* programs can be intuitively modeled as web services in the scientific workflow framework in a graphical manner [8]. In *LCDL*, web service interfaces are designed graphically using the *Eclipse Modeling Framework*. Data transfer between *LCDL*-generated web services is realized with the *Flex-SwA* middleware [6]. With *LCDL* and *Flex-SwA*, the integration of

existing *13CFLUX2* programs into the scientific workflow architecture is easily possible.

5 Conclusions

In the contribution we sketched the implementation of the metabolic network modeling workflow within our scientific workflow framework. The modeling workflow basically consists of the following four interlinked steps: (1) visual modeling of a metabolic reaction network, (2) mathematical modeling of reactions, atom transitions and biochemical constraints, (3) evaluation of experimental data sets, i.e. the estimation of model parameters, and (4) visualization of temporary and final results. Data integration and dynamic human tasks are involved in all steps of the 13C-MFA workflow.

The presented scientific workflow framework approaches the integration of human tasks and knowledge sources by recombining well-established software tools. While leaving the core software untouched, *Omix* was extended by a flexible plug-in interface, allowing user-oriented integration of scientific knowledge sources. The integration of the high-performance simulation toolbox *13CFLUX2* into a SOA was performed by wrapping each program with a simple web service interface. Thus, these programs are seamlessly integrated into the workflow framework, allowing an intuitive usage of scientific 13C-MFA applications. The graphical web service modeling framework *LCDL* and the *Flex-SwA* data exchange middleware are used for the integration of *13CFLUX2* programs into a SOA environment. These tools have been proven to work well together in combination with the workflow orchestration software *ActiveBPEL* [8,2].

The presented modeling workflow was implemented aiming at liberating scientists from organizational and recurring tasks. Basic knowledge acquisition, routine data conversion and processing steps are handled by the scientific workflow framework. This software-aided treatment not only reduces complexity of 13C-MFA applications, but also helps to avoid manual errors. The major aim of the scientific workflow framework is automation and, in turn, acceleration of the overall processing time. At the end, this paves the way for a reproducible handling of higher-throughput data.

Future work in this area includes the implementation of a 13C-MFA database system, offering models, experimental data and simulation results. Major challenges for a 13C-MFA database middleware are user management and access control. After augmenting all *13CFLUX2* programs with web service functionality, the development of further 13C-MFA scientific workflows with automation and Grid provisioning can be easily achieved.

Acknowledgments. We would like to thank Ernst Juhnke, Tim Dörnemann and Bernd Freisleben (Dept. of Mathematics and Computer Science, University of Marburg) for their support with *LCDL*, *Flex-SwA* and *ActiveBPEL*. We thank Tobias Vehrkamp for implementing the KEGG plug-in.

References

1. Barseghian, D., Altintas, I., Jones, M.B., Crawl, D., Potter, N., Gallagher, J., Cornillon, P., Schildhauer, M., Borer, E.T., Seabloom, E.W., Hosseini, P.R.: Workflows and extensions to the kepler scientific workflow system to support environmental sensor data access and analysis. Ecological Informatics 5(1), 42–50 (2010)
2. Dörnemann, T., Juhnke, E., Freisleben, B.: On-Demand Resource Provisioning for BPEL Workflows Using Amazon's Elastic Compute Cloud. In: Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2009), pp. 140–147. IEEE Press, Los Alamitos (2009)
3. Droste, P., von Lieres, E., Wiechert, W., Nöh, K.: Customizable Visualization on Demand for Hierarchically Organized Information in Biochemical Networks. In: Barneva, R.P., Brimkov, V.E., Hauptman, H.A., Natal Jorge, R.M., Tavares, J.M.R.S. (eds.) CompIMAGE 2010. LNCS, vol. 6026, pp. 163–174. Springer, Heidelberg (2010)
4. Droste, P., Noack, S., Nöh, K., Wiechert, W.: Customizable Visualization of Multiomics Data in the Context of Biochemical Networks, pp. 21–25. IEEE Computer Society, Los Alamitos (2009)
5. Droste, P., Weitzel, M., Wiechert, W.: Visual exploration of isotope labeling networks in 3D. Bioprocess and Biosystems Engineering 31, 227–239 (2007)
6. Heinzl, S., Mathes, M., Friese, T., Smith, M., Freisleben, B.: Flex-SwA: Flexible Exchange of Binary Data Based on SOAP Messages with Attachments. In: Proc. of the IEEE International Conference on Web Services, Chicago, USA, pp. 3–10. IEEE Press, Los Alamitos (2006)
7. Hull, D., Wolstencroft, K., Stevens, R., Goble, C., Pocock, M., Li, P., Oinn, T.: Taverna: a tool for building and running workflows of services. Nucleic Acids Research 34, 729–732 (2006)
8. Juhnke, E., Seiler, D., Stadelmann, T., Dörnemann, T., Freisleben, B.: LCDL: An Extensible Framework for Wrapping Legacy Code. In: Proceedings of ERPAS 2009, pp. 646–650 (2009)
9. Juhnke, E., Dörnemann, T., Freisleben, B.: Fault-Tolerant BPEL Workflow Execution via Cloud-Aware Recovery Policies. In: Software Engineering And Advanced Applications, SEAA 2009, pp. 31–38 (2009)
10. Kitano, H.: Systems Biology: A Brief Overview. Science 295(5560), 1662–1664 (2002)
11. Lamprecht, A.L., Margaria, T., Steffen, B.: Bio-jETI: a framework for semantics-based service composition. BMC Bioinformatics 10(suppl. 10), S8 (2009)
12. Lamprecht, A.L., Margaria, T., Steffen, B., Sczyrba, A., Hartmeier, S., Giegerich, R.: GeneFisher-P: variations of GeneFisher as processes in Bio-jETI. BMC Bioinformatics 9(suppl. 4), S13 (2008)
13. Ludäscher, B., Altintas, I., Berkley, C., Higgins, D., Jaeger, E., Jones, M., Lee, E.A., Tao, J., Zhao, Y.: Scientific workflow management and the Kepler system. Concurrency and Computation: Practice and Experience 18(10), 1039–1065 (2006)
14. Mietzner, R., Karastoyanova, D., Leymann, F.: Business Grid: Combining Web Services and the Grid. In: Jensen, K., van der Aalst, W.M.P. (eds.) Transactions on Petri Nets and Other Models of Concurrency II. LNCS, vol. 5460, pp. 136–151. Springer, Heidelberg (2009)
15. Möllney, M., Wiechert, W., Kownatzki, D., de Graaf, A.A.: Bidirectional reaction steps in metabolic networks,: Optimal design of isotopomer labeling experiments. Biotechnology and Bioengineering 66(2), 86–103 (1999)

16. Neuweger, H., Albaum, S.P., Dondrup, M., Persicke, M., Watt, T., Niehaus, K., Stoye, J., Goesmann, A.: Meltdb: a software platform for the analysis and integration of metabolomics experiment data. Bioinformatics 24(23), 2726–2732 (2008)
17. Ngu, A.H., Bowers, S., Haasch, N., Mcphillips, T., Critchlow, T.: Flexible Scientific Workflow Modeling Using Frames, Templates, and Dynamic Embedding. In: Ludäscher, B., Mamoulis, N. (eds.) SSDBM 2008. LNCS, vol. 5069, pp. 566–572. Springer, Heidelberg (2008)
18. Nielsen, J., Jewett, M.C.: Impact of systems biology on metabolic engineering of Saccharomyces cerevisiae. FEMS Yeast Research 8(1), 122–131 (2008)
19. Shoshani, A., Rotem, D.: Scientific Data Management: Challenges, Technology, and Deployment. Chapman & Hall/CRC (2009)
20. Tan, W., Missier, P., Foster, I., Madduri, R., De Roure, D., Goble, C.: A Comparison of Using Taverna and BPEL in Building Scientific Workflows: the case of caGrid. Concurr. Comput.: Pract. Exper. 22(9), 1098–1117 (2010)
21. Tang, Y.J., Martin, H.G., Myers, S., Rodriguez, S., Baidoo, E.E.K., Keasling, J.D.: Advances in analysis of microbial metabolic fluxes via ^{13}C isotopic labeling. Mass Spectrometry Reviews 28(2), 362–375 (2009)
22. Taylor, I.J., Deelman, E., Gannon, D.B., Shields, M.: Workflows for e-Science: Scientific Workflows for Grids. Springer, New York (2006)
23. Wächter, A.: An Interior Point Algorithm for Large-Scale Nonlinear Optimization with Applications in Process Engineering. Ph.D. thesis, Carnegie Mellon University (2002)
24. Wang, J., Crawl, D., Altintas, I.: Kepler + Hadoop: A General Architecture Facilitating Data-Intensive Applications in Scientific Workflow Systems. In: WORKS 2009: Proceedings of the 4th Workshop on Workflows in Support of Large-Scale Science, pp. 1–8. ACM, New York (2009)
25. Weitzel, M.: High Performance Algorithms for Metabolic Flux Analysis. Ph.D. thesis, University of Siegen, Germany (2009)
26. Wiechert, W.: ^{13}C Metabolic Flux Analysis. Metababolic Engineering 3(3), 195–206 (2001)
27. Wiechert, W., Siefke, C., de Graaf, A.A., Marx, A.: Bidirectional reaction steps in metabolic networks, Part II: Flux estimation and statistical analysis. Biotechnology and Bioengineering 55(1), 118–135 (1997)
28. Zamboni, N., Fendt, S.M., Rühl, M., Sauer, U.: ^{13}C-based metabolic flux analysis. Nature Protocols 4(6), 878–892 (2009)

Intelligent Document Routing as a First Step towards Workflow Automation: A Case Study Implemented in SQL

Carlos Soares[1,2] and Miguel Calejo[3,4]

[1] LIAAD-INESC Porto LA, Univ. of Porto, R. Ceuta, 118, 6., 4050-190 Porto, Portugal
[2] Faculdade de Economia, Universidade do Porto
[3] Declarativa
[4] Information Systems Department, Universidade do Minho, Portugal
csoares@fep.up.pt, mc@declarativa.pt
http://www.declarativa.pt/

Abstract. In large and complex organizations, the development of workflow automation projects is hard. In some cases, a first important step in that direction is the automation of the routing of incoming documents. In this paper, we describe a project to develop a system for the first routing of incoming letters to the right department within a large, public portuguese institution. We followed a data mining approach, where data representing previous routings were analyzed to obtain a model that can be used to route future documents. The approach followed was strongly influenced by some of the limitations imposed by the customer: the budget available was small and the solution should be developed in SQL to facilitate integration with the existing system. The system developed was able to obtain satisfactory results. However, as in any Data Mining project, most of the effort was dedicated to activities other than modelling (e.g., data preparation), which means that there is still plenty of room for improvement.

1 Introduction

In spite of the maturity of Information Systems (IS) and the central role that they play in organizations today, they have not eliminated the need for paper [7]. In fact, in many cases they have increased the amount of paper circulating within and between organizations. In many cases, they have also failed to improve the efficiency of the processes of organizations. The area of workflow automation addresses these problems. Its goal is "to automate business processes by coordinating and controlling the flow of work and information between participants" in those processes [7]. Note that the term "participants" includes not only members of the organization but also external partners (e.g., customers and suppliers).

The automation of workflows cannot be achieved simply by installing some software for that purpose. Most of the time, it requires managerial changes which

T. Margaria and B. Steffen (Eds.): ISoLA 2010, Part I, LNCS 6415, pp. 276–284, 2010.

are difficult. The difficulties in making these changes (or, alternatively, ignoring the need to make them) increases the probability of failure of workflow automation projects [7], especially in large and complex organizations.

An easier task is the elimination of paper. Many public and private institutions have significantly reduced the amount of paper in their operations, particularly for internal documents. Elimination of external documents, both incoming and outgoing, is harder (e.g., authentication). However, having documents in digital form does not eliminate the need to route them to the right people who can process them. This issue is particularly important in the moment that external documents enter the organization. Sending a document to the wrong person will delay its processing, reducing the efficiency of the organization and possibly causing inconveniences to their customers, suppliers and other partners. However, given that the documents are in digital form, it is possible to develop software to automate the process of routing them [5,3]. In many cases, this is a first step towards workflow automation that raises less resistance while enabling important gains in the efficiency and efficacy of the processes of the organization.

Our case study is concerned with routing of incoming documents in a large public institution. This institution has a document management system. When documents arrive, they are digitized and inserted into this system by a human operator. This operator then routes the document to one of the many departments of the institution. The document may require further routings within the same department or to different departments. However, the first routing is critical because it is done by an operator who is moderately skilled and may forward the document incorrectly, which reduces the efficiency of the organization, as discussed earlier. Further routings are typically done by employees who are more familiar with the process that the document is associated with and, thus, have less probability of error. Additionally, the operator must read the document, even if superficially. In an institution with a large volume of correspondence, the amount of effort required for this operation is significant.

This project addresses the problem of providing automatic support to the first routing with the goal of making it a more efficient and eventually less error-prone process. The aim is to develop a model to make recommendations to the operator concerning where to route the document to. We note that the routing is not fully automatic, as the operator is always given the possibility to override it.

A data mining approach is followed. Historical data of previously made routings are used to obtain a model that relates characteristics of documents to their destination. However, this project was shaped by important constraints. Firstly, the budget is rather small. Only a few persons/days are available. Due to the uncertain nature of data mining projects, which will only be successful if the data available contains the necessary information, this kind of short-term, preliminary study is becoming increasingly popular, possibly also due to the current economical context. A second constraint is technical. Although the project could be developed with any tool, the solution must be easy to integrate into the software. Ideally, it should be in SQL, as this is the main technology of the workflow and document management software used by this institution.

In the remainder of the paper we describe the project, loosely following the CRISP-DM methodology, which was, again loosely, followed in this project [1]. In Section 2 we describe the business and data understanding phases. Next, we discuss the preprocessing operations required to prepare the data for mining (Section 3). In Section 4 we describe the modelling phase as well as the results, and the implementation is described in Section 5. We finish with some conclusions in Section 6.

2 Business and Data Understanding

It is well known that these two phases are essential to the success of a data mining project [1,4]. If they are not given sufficient attention, the user may end up looking for the answers to the wrong questions. Therefore, they usually take a significant amount of the time allocated to DM projects. However, due to the small size of the project described here, a very limited amount of time was dedicated to them. Essentially, the activities carried out were:

- a couple of initial meetings to specify the problem and establish the goals,
- reading software documentation, focusing particularly on the description of the schema of the database that contains the data to be used in the project, and
- a few dozen emails and phone calls to clarify remaining issues.

One important issue is that it was not possible, due to the constraints of the project, to quantify business or data mining goals. There was a common agreement regarding the impossibility of doing so in a short project. Therefore, it was decided to establish the following objectives: simply prove that it is possible to implement a feasible solution for the problem, technically and in terms of the quality of the routings recommended by the model (i.e., the model is doing better than a suitable baseline).

As mentioned earlier, the model was obtained using historical data. This data was available in a database that supports the document management system of the organization. It contains more than 20 tables, storing information about more than 600,000 documents. Much of the information in this database is not relevant for this work, so the first step was to identify relevant information.

In particular, the database may contain many routings for each document. Since the object of analysis in this project is the first routing of a document, the relevant subset of routings was determined. This is not as easy as it may seem due to issue concerning the schema of the database. Thus, given the limited time available and the complexity of the database, it was very important to do this phase in very close collaboration with both the domain experts and the developers of the software. After a few attempts it was decided that the developers of the software should provide a query to identify the set of first routings. The complexity of the database is illustrated by the fact that this query combined data from five different tables.

No matter how carefully this step is carried out, there is a high probability that the set of objects selected for analysis is incorrect, either because it contains

Table 1. Number of documents in the various iterations of the process

Iteration	Number of documents
0 (original database)	621,587
1	116,757
2	153,614
2 (containing references to legislation)	15,331

objects which should not be selected or because it is missing some which should have been selected. This project was no exception to this rule. The first definition of first routings given by the domain experts was incorrect. This problem was identified after some time and a new query was provided representing the correct definition. As expected by the domain experts, the selection of first routings for analysis significantly reduced the number of documents to be handled in the project (Table 1). The results presented here refer to the second iteration, except when stated otherwise.

Due to the large number of tables in the database and the complexity of its organization, it was important to identify a small number of variables that are expected to contain information that is useful in predicting the first routing of a document. According to the domain experts, two types of information are expected to be useful:

references to legislation the documents contain references to laws and other kinds of official documents. These references tend to contain a lot of information about the subject of the document and, consequently, about who should process it. Recently the document management software was enhanced with capabilities to detect these references, so they can be used as predictors of the routing destination.

subject the subject of the document is also expected to contain information about its purpose. However, as it is free text, it is much noisier.

By focusing on these variables, we were able to add only one more table as a source of data. As can be seen in Table 1, given that the identification of references to legislation is quite recent, the number of documents for analysis is significantly reduced. On the other hand, given the limited time available, the analysis based on the subject was only carried out with the documents selected in the first iteration (i.e., with the earliest definition of first routing which ended up to be incorrect).

Finally, the target variable was identified as the department where the document was first routed to. To determine the target variable it was necessary to use three more tables. This means that only nine of the tables were used in total. This reduction in the number of tables that must be dealt with is essential in a small project such as this one. This is unike what would happen in a larger project. In the latter case, the focus would be on using as much useful information as possible, requiring much more work to collect it.

3 Preprocessing and Exploratory Data Analysis

After selecting the data it is very important to get familiar with it and also to prepare it for the modeling phase. Therefore a significant proportion of project time was dedicated to this task. One particularly important goal of this phase is to assess the quality of the data and, if necessary, correct it. The exploration of the data typically consists of generating a large number of statistics and plots and then analyzing more carefully the most relevant observations. Here we will discuss only a few of the observations made.

Data exploration is usually done using a sample of the data. However, the limited time available for the project forced us to work with the full data immediately. This raised important challenges in terms of the computational efficiency of the tools, which must be very high.

Table 2 shows some basic statistics. As can be observed, only 20 of the 56 departments have 100 or more routings. This means that for the majority of the departments (36), the number of documents available to identify routing patterns is very small. Therefore, using the current set of data, it is expected to be very difficult to obtain accurate routing models for these departments.

Besides the target variable, it is also very important to explore the information in the predictor (independent) variables. Some statistics concerning the references to legislation in the documents are presented in Tables 3 and 4.

Note that, as stated earlier, the number of documents that are represented in the database having references to legislation is very small because this funcionality was implemented only very recently.

Several data quality problems were identified in this analysis. Table 3 illustrates one of them: there are three documents with almost more than 100 references to legislation. This may be due to errors in the automatic process of identifying those references. Even if it is not, these documents represent outliers, which may affect the models obtained.

Table 2. Class frequency: number of departments with the given number of first routings

Routed documents	Departments
1 or more	56
20 or more	35
100 or more	20

Table 3. Frequency of references to legislation per document: number of documents with the number of references indicated

Number of references to legislation	Documents
1	7,266
10 or more	669
100 or more	3

Table 4. Documents per law (or other official document): number of laws referred in the given number of documents

Number of documents	Number of laws (or other official document)
1 or more	8,779
50 or more	149
100 or more	59

Given the similarities of this problem with the document classification problem, the representation of the documents was based on the bag-of-words concept [6], as is usual in text mining and information retrieval. Each document is, thus, represented as a binary vector. Each element of a vector represents a law (or other official document) or, in the second type of predictor variables, a term in the subject. The binary vector represent the presence or absence of the reference/term in the document.

4 Modeling and Results

According to the domain experts, in many cases, the references to legislation determined the department to which the document should be routed. Therefore, we started with a very simple mode based on conditional probability tables, as follows:

$$p_{i,j} = p(dep_i|ref_j) \tag{1}$$

in which dep_i is the department i and ref_j is law (or official document) j. A routing rule is created for each $p_{i,j} > t$, where t is a given threshold. This rule means that a document containing a reference to ref_j should be routed to department d_i.

Therefore, for each new document, the routing rules that apply are determined given the references to legislation in that document. This process may yield the following outcomes:

unique routing the applicable rule(s) route the document to a single department. In this case, a routing to that department is recommended.

conflict several rules apply and some point to different departments. In this case, a meta-rule to solve these conflicts is necessary. This could be, for instance, the department, among those that are recommended, with the highest prior probability, $p(i)$, or the routing suggested by the routing with the highest probability.

no routing none of the rules apply. In this case, it is necessary to have a default routing (i.e., the department with the highest prior probability, $p(i)$), the operator is asked to do the routing manually or rules with a probability smaller than the established threshold t could be applied.

A suitable methodology is required in order to evaluate the model. In this case, a hold-out strategy was followed. The set of documents was split into two subsets, training and test, with approximately 70% and 30% of the documents, respectively. Table 5 presents the results on the test set.

Table 5. Results obtained with the routing rules

number of documents	4,600 (30%)
unique routing	1,080 (24%)
accuracy of unique routing	67%
conflicts	77

Besides this approach, which was only tried with references to legislation, a series of other common machine learning and statistical algorithms were also tested with the two representations (references to legislation and subject), including naive Bayes, decision trees, support vector machines and random forests. However, no significant improvements were obtained and the computational cost was much higher. Additionally, their integration with the document management system was technically more complex.

5 Implementation

During the development of the project, the following tools were used:

- SQL Server Express 2005 running on an HP server
- R 2.9.0 on a Macbook with an Intel Core duo 2 GHz, 4G of memory and running Mac OS X (10.4)
- SQL Server Management Studio Express on HP Compaq 8000 Intel Core duo 2 GHz, 4G of memory and running Windows XP
- SQL Server Management Studio Express on HP running Windows XP

One of the constraints of the project, as mentioned earlier, was to implement the solution in SQL. This implementation proved to be quite easy, involving:

- 2 new tables
- 3 views for modeling
- 7 views for recommendation

The solution proved also to be quite efficient. Execution on the system indicated above took less than 10 minutes, including the time to build the dataset from the set of original 600,000 documents.

We note that the deployment phase is critical in most data mining projects. Many projects with good modeling results fail because the solution developed is not adequately integrated in the existing information systems. This helps explain our concern with the technical restrictions of the project.

6 Conclusions and Future Work

This paper described a small project to develop a system for supporting document routing in a public institution, more specifically, the recommendation of

the right department within the institution of which incoming correspondence should be sent.

A data mining approach was followed, where data representing previous routings were analyzed to obtain a model that can be used to route future documents. The approach followed was largely influenced by some of the limitations imposed by the customer: the budget available was quite small and a SQL-based solution was required, to facilitate integration with the existing system. A very simple solution was devised and successfully deployed.

The work carried out is summarized in Table 6.

Table 6. Summary of results

Documents		
	All	621,587
	Selected	15,331
	Test	4,600
	Training	10,731
Departments (target variable)		
	All	56
	With 100 or more documents	20
Representation	binary (*bag-of-words* style)	
Experimental methodology	*hold-out* 70/30	
Algorithm	simple conditional probabilities	
Results		
	routed documents	1,080 (24%)
	... with accuracy	67%
	conflicts	77 (2%)

From an implementation perspective, the project was a success: the solution was easily integrated with the existing system and it is very computationally efficient. In terms of the quality of the recommendations, the results were as satisfactory as could be expected from a project with such strong limitations, mainly in terms of human resources. Some of the data preparation and modeling issues that should be addressed in a follow-up project are:

- explore the database further, in order to identify more variables with predictive ability;
- address the problem of unbalanced class distribution [2];
- complete the data cleaning step, namely by assessing how the quality of the routings in the data (i.e., are all routings represented in the data correct);
- explore other document classification approaches [6].

In terms of implementation, further work is equally necessary. First of all, it is necessary to implement a maintenance mechanism. This implies recording every recommendation made by the model and the routing made by the operator, if the recommendation is not followed. This information, together with further

routing data, can be used to monitor the quality of the model. Additionally, it would be interesting to test the data mining primitives that are implemented in SQL Server.

This work is also a first step in the development of a workflow automation system for the organization. The success of the solution described here is expected to make people more open to a larger workflow automation project, which will probably imply some changes to their work processes.

This project also raises some interesting issues concerning the development of low-budget data mining projects. The first one concerns the methodology. CRISP-DM is designed for medium to large-scale projects. It is flexible enough to be adapted for smaller projects, as shown here. However, it would be important to have a simplified version of the methodology, which helps the data miner focus on the essentials. Three issues we found very important are:

customer participation it is essential that the customer has some experience on participation in data mining projects. If there is some knowledge of data mining in the customer, this is even better.

data readiness if the previous item is true, then it should be possible that the customer may provide you with a version of the data which is almost ready for data mining. This means, representing the correct population of objects to be analyzed, including the most important predictor variables and being minimally clean. The effort required to make the data ready for the data mining tools should be minimal.

experimental setup readiness after the preparation of the data, as described in the previous issue, modeling should mostly consist of pushing a button and then analyzing results. The issue of efficiency of the tools is very important in this context as well. We observed that database management systems (DBMS) are one possibility that should be considered.

References

1. Chapman, P., Clinton, J., Kerber, R., Khabaza, T., Reinartz, T., Shearer, C., Wirth, R.: CRISP-DM 1.0: Step-by-Step Data Mining Guide. In: SPSS (2000)
2. Chawla, N.V., Japkowicz, N., Kotcz, A.: Editorial: special issue on learning from imbalanced data sets. SIGKDD Explor. Newsl. 6(1), 1–6 (2004)
3. Cheng, I., Srinivasan, S., Boyette, N.: Exploiting XML technologies for intelligent document routing. In: DocEng 2005: Proceedings of the 2005 ACM Symposium on Document Engineering, pp. 26–28. ACM, New York (2005)
4. Frawley, W.J., Piatetsky-Shapiro, G., Matheus, C.J.: Knowledge discovery in databases: an overview. AI Mag. 13(3), 57–70 (1992)
5. Krishna, V., Deshpande, P.M., Srinivasan, S.: Towards smarter documents. In: CIKM 2004: Proceedings of the Thirteenth ACM International Conference on Information and Knowledge Management, pp. 634–641. ACM, New York (2004)
6. Sebastiani, F.: Machine learning in automated text categorization. ACM Comput. Surv. 34(1), 1–47 (2002)
7. Stohr, E.A., Zhao, J.L.: Workflow automation: Overview and research issues. Information Systems Frontiers 3(3), 281–296 (2001)

Combining Subgroup Discovery and Permutation Testing to Reduce Reduncancy

Jeroen S. de Bruin[1,2] and Joost N. Kok[1]

[1] LIACS, Leiden University, Niels Bohrweg 1, 2333 CA Leiden, The Netherlands
[2] LUMC, Biomolecular Mass Spectrometry unit, Department of Parasitology, Einthovenweg 20, Postbus 9600, 2300 RC Leiden, The Netherlands

Abstract. Scientific workflows are becoming more popular in the research community, due to their ease of creation and use, and because of the benefits of repeatability of such workflows. In this paper we investigate the benefits of workflows in a genomics experiment which requires intensive computing as well as parallelization, and show that substantial optimizations in rule redundancy reduction can be achieved by simple workflow parallelization.

1 Introduction

Over the past few years, scientific workflows became a popular topic of research, especially in areas that deal with computationally-intensive, repetitive processes such as genomics and proteomics. We define a *workflow* as a collection of components and relations among them, together constituting a process. Components in a workflow are entities of processing or data. They are connected by relations, which can either be data transport entities that connects inputs and outputs from one component to another, or control flow entities that impose conditions on the execution of a component.

Due to the increased popularity of workflows, workflow editors and frameworks have become increasingly popular over the last few years, since they enable a scientist to graphically construct a process of interconnected building blocks, allowing for easier experiment design and easier use of distributed resources. Taverna [17] is an example of a workflow designer that allows for easy creation of workflows, possibly with remote resources.

When performing experiments within the fields of genomics or proteomics, e.g. microarray or SNP experiments for gene expression, or FTICR protein identification experiments, the outcome is usually an identifier of a gene or peptide accompanied by a certain score or indication of likelyhood. Interpreting such a list is often hard, sometimes because of the sheer number of the identifiers, or because relations between identifiers are not so clear or straightforward.

A way to make relations between concepts or identifiers more clear is through the use of ontologies. An ontology, as seen in information science, is the hierarchical structuring of knowledge about things by subcategorizing them according to their essential (or at least relevant and/or cognitive) qualities [19]. Over time, many efforts have been made by the computer science community together with the bio-informatics community to create ontologies in order to have a common reasoning platform [3,18].

T. Margaria and B. Steffen (Eds.): ISoLA 2010, Part I, LNCS 6415, pp. 285–300, 2010.

When generating rules from a ranked list of identifiers, the rules usually reflect the ranking. It is therefore good practise, where possible, to make sure that the ranking is correct (or at least plausible), and to make sure that rules generated and reported are specific to that ranking, and not a product of randomness or chance, which can sometimes occur. We can do this by using a variation on Fisher's Exact Test [8,9], which generates permutations of the original input. By generating permutations of a ranked list, one can check if these permutations generate similar rules. If so, then the rules become less important and interesting, for they are not specific to the original ranking.

We propose a service called *Fantom* that uses ontologies to uncover and clarify relations between identifiers in an experiment. The service mines all interesting subgroups and describes them by a conjunction of ontological predicates. The service is designed in a generic way so that we can apply it in a variety of fields. As input, it uses a set of identifiers coupled to a set of scores and allowed ontological predicates. As output, it presents the user with a set of rules and an appreciation of those rules in the form of a score measure. Furthermore, we also apply input permutation to the rules generated in Fantom to improve rule pruning and threshold selection.

This paper is organized as follows. In Section 2, we will discuss work related to our Fantom approach, and discuss various knowledge sources that are used in Fantom as well. In Section 3, we will discuss Fantom itself, providing a detailed overview of inputs and outputs, rule generation and rule pruning algorithms. In Section 4, we discuss exact testing with Fantom on a single-class problem, generating a rule list with rules unique to the original permutation. We also discuss exact testing with Fantom for multi-class problems, whereby different groups of identifiers are compared to each other. We explain the difference with the normal experimental setup, provide detailed workflows, and present the algorithm behind automatic thresholding of different rule participants. In Section 5, we present experimental results on both variations using a gene expression dataset. Finally, in Section 6 we present some conclusions and future work.

2 Related Work

In this section we present work related to the Fantom service, as well as work related to structured knowledge sources, knowledge mappings and other sources of information used in Fantom.

2.1 Ontologies

Due to the increased focus on data mining with ontologies, related technologies such as representations of ontologies, description logic and ontology reasoning have been given much attention as well. Currently, there is a wide range of (ontology) description languages available, and each of them has their own specific role. For representation of ontology elements and data, usually a form of the XML is applied, sometimes together with the Resource Description Framework (RDF) [25]. For representation of relations among the data elements and extensions to allow reasoning over these entity-relationship models, currently the Web Ontology Language (OWL) [26] and the older

F-Logic [4] are commonly used. Within bio-informatics, the Gene Ontology (GO) [3] and the Kyoto Encyclopedia of Genes and Genomes (KEGG) [18] are widely known. A good overview of ontology languages is provided in [21].

2.2 Annotations and Mappings

Within Fantom we use ontological terms as rule predicates. This would not be possible if a mapping that associates or correlates an element with one or more ontology terms was not available. In the field of bio-informatics, there are many identifiers that can be used in genomics and proteomics [15,20,16,27] and they are usually accompanied by mappings between those identifiers and ontologies, or those identifiers and other identifiers.

Fantom provides an option to generate interaction association rules. By using a data source that states interactions between identifiers, Fantom can uncover patterns that describe these indirect relations. These interaction patterns have the form of "interacts with (rule)". In genomics, interactions can for example be obtained from the GeneRIF project [5] and Reactome [24].

2.3 Related Algorithms

The Fantom algorithm is based on the SEGS algorithm described in [22] and its predecessor [23]. While SEGS uses a similar method to Fantom, it is restricted to only one entry per (sub)ontology, is tailored specifically to microarray experiments, and does not prune rules that provide redundant information, nor does it provide clustering of similar rules.

In [14] subgroups are matched to a subset of GO terms in a probabilistic way, which induces a greater portion of error and false discoveries than an exhaustive search through the search-space. The GOEAST [28] algorithm also checks for gene enrichment in GO terms, but only checks for a single GO term, and takes as input raw microarray data, which again restricts its applicability.

Another tool that mines gene lists is DAVID [12], the Database for Annotation, Visualization and Integrated Discovery. Its functionality is similar to Fantom: it can classify large gene lists into functional related gene groups by relating them to ontologies (so far only the GO ontology was seen in the outputs), rank the importance of the discovered gene groups and summarize the major biology of the discovered gene groups. It also has capabilities to visualize genes and their functional annotations in a group.

The main difference is that DAVID does not allow for scored lists of genes. It solely acts on the genes that are entered in a list, and thus treats each gene as equally important. Furthermore, the number of genes allowed to experiment on is restricted to 3000, which is not much considering microarray experiments can easily comprise tens of thousands of genes. Furthermore, rule mining based on interaction associations is not available, and clusterering is done by using fuzzy heuristic partitioning instead of a similarity measurement. Finally, DAVID is not available as a web service but as a website, making it more complex to integrate in modern workflow designer tools.

A lot of work has also been done on scoring functions. Typically, there is not just one scoring function that is considered the best, it all depends on what research is being conducted and what properties are considered interesting. In the case of Fantom, the aim

is to have a score for a subset of elements from a ranking of identifiers and scores; the group score is thus dependent on the score of individual elements. In bio-informatics, well-known algorithms that perform this kind of scoring are found in [10,13].

3 The Fantom Service

Fantom is a service that relates subgroups of identifiers to ontological knowledge that these subgroups have in common, or to ontological descriptions of identifier groups that they interact with. It takes as input a set of identifiers and their scores, background knowledge in the form of ontologies, mappings and interaction data, a scoring function, and thresholds for rule generation, and through rule generation and pruning delivers a non-redundant set of rules that describe subgroups of the input set. In this section we will discuss the most important aspects of Fantom, such as the ontologies, mapping, output, scoring, rule generation and rule pruning.

3.1 Ontologies

The Fantom service aims to find groups of identifiers that relate to a conjunction of terms within an established knowledge base. We use ontologies as that knowledge base. An ontology is a hierarchical structuring of knowledge, whereby broader, more general terms form the root of the ontology, and subsequent specifications and differentiations of those root terms form the children. We call these ontological terms *concepts*.

In the ontologies used in Fantom we place a restriction, namely that they must be organized in a directed acyclic graph, whereby each connection between concepts has a specialization (”is a”) or aggregation (”part of”) relationship.

3.2 Mapping

Mappings and interactions are used to relate identifiers to associated concepts, or to relate identifiers to other identifiers. In the Fantom service we strive to keep one identifier class central, and use mappings to map the central class to other classes, and vice versa. For example, in case of our bio-informatics experiments, we keep mappings of ENTREZ [15] gene identifiers to ontologies like GO and KEGG, and use mappings from SYMBOL [27] to ENTREZ in order to translate the input ranking and output rules. The same is true for interaction definitions; all interaction between genes that were found in [5] and Reactome [24] are expressed by ENTREZ identifiers.

3.3 Scoring

The scoring functions are those functions that take as input a subset of identifiers corresponding to a rule along with their individual scores, and have as output a single numeric value indicating the *interestingness* of the rule, a value that lies between 0 and 1, 0 being most uninteresting and 1 being most interesting. By default, the Enrichment Score (ES) function [10] is selected, which calculates the score of a subgroup based on the score of its individual members as well as the members not in the subgroup.

Despite the fact that this measurement was devised to express the interestingness of gene sets, we think that it is in no way restricted to that purpose. We argue that any ranked set with a individual scores that indicate correlation or effect with respect to a certain experiment can be used in this scoring measurement.

3.4 Output

As output, Fantom generates a text file that contains all the rules that remain after pruning. Furthermore, rules with the same subset of identifiers are clustered together, improving readability. An example rule looks like this:

```
Rule 1
Score: 0,761
Participants: [Epha1, Epha2, Epha8, Ephb2, Ephb3, Ephb4]

All genes in the subgroup
 have the following properties:
  molecular_function(protein tyrosine kinase activity),
  molecular_function(ATP binding),
  biological_process(protein amino acid phosphorylation),
  biological_process(tyrosine kinase signaling pathway),
  KEGG_pathway(Axon guidance)
```

The example rule describes a certain subgroup containing the genes *Epha1, Epha2, Epha8, Ephb2, Ephb3, Ephb4* and relates them to GO and KEGG terms, a process we call *Knowledge Fitting*. By relating subgroups of the input to established knowledge sources, we strive to increase the interpretability of knowledge.

3.5 Rule Generation

Rule generation in the Fantom service is based on the Apriori algorithm, which is frequently used in itemset mining [1] and refined many times since its conception [11,6]. Within the Fantom service, the algorithm repeatedly executes three stages: candidate generation, candidate appreciation and candidate pruning. A central structure that manages rules in all these procedures is the *SubsetCollection* structure, which provides methods for inserting, retrieving and discarding rules.

A rule is a data structure with two lists; one list contains the identifiers that participate iin the rule, while the second list contains the ontological concepts that form the rule. Rule generation proceeds on the basis of subset combination; two rules that contain a conjunction of m concepts are combined with each other to form a rule that has a conjunction of $m + 1$ concepts. However, if the two concepts that are combined are hierarchically related, then the combination is invalidated, and the rule is discarded.

3.6 Rule Pruning

Rule pruning is done to make sure only the most interesting rules are reported back once the experiment is over, to make sure redundant information is left out, but also

to minimize the amount of rules generated in the experiment. Within each stage, rule pruning is done in three phases. First, rules generated and appreciated in the previous two phases are now compared to the input constraints; all rules whose number of participants do not satisfy the support or score constraints are pruned. The remainder will be inserted in the SubsetCollection structure, where further pruning will take place.

Within the SubsetCollection structure, pruning proceeds in a bi-dimensional way, hence we called it *Bi-Cohortal Pruning*. First we prune the rules in a *horizontal* cohort, comparing rules of the same dimensionality (by dimensionality we mean the number of concepts that an rule contains). The remaining rules are inserted one by one.

For every rule containing $m > 1$ ontological concepts, m subgroups are created. For each subgroup, the rule is compared to other rules of the same dimensionality for redundant knowledge, checking whether their differentiating concepts (the two concepts these rules do not have in common) are related. If this is the case, and the score of the more specific rule is higher than or equal to the score of the more general one, then the more general rule is discarded as a rule (but is still kept for future rule generation, and inserted in all the subgroups it belongs to).

After pruning of the horizontal cohort has finished, pruning of the *longitudinal* cohort commences. The remainder of the still valid rules of dimensionality m are compared to those of dimensionality $1...m-1$, since rules of dimensionality m are more specific than rules of a lesser dimensionality, and can thus render those rules obsolete. To this end, all subsets of dimensionality $1...m-1$ are generated from each remaining rule of dimensionality m, and these are subgroups are compared to in the subset collection by comparing them to related entries in the castable of their respective dimensionality.

4 Exact Testing for Pruning and Optimization

In this section we discuss exact testing with Fantom to optimize pruning and automatic optimal theshold determination through the use of workflows and paralellism that are inherent to the workflows.

4.1 Exact Testing: Single-Class Pruning Optimization

In this variant of Fantom, we use the Fantom algorithm repeatedly to generate rules from multiple identifier lists. One list is the original ranking, which reflects the original experiment data, and the other lists are the results of permutations of that data or ranking. It is up to the experimenter to create those permutations, although we created a permutation algorithm for microarray expression data.

Exact testing for single-class pruning is a three-stage process, as shown in Figure 1. In the first stage, input permutations are generated. The output of this phase, a set of ranked lists, will be presented to the second stage, along with the original ranked list.

The second stage is the concurrent execution of the Fantom service on all ranked lists generated in phase one. Depending on how many permutations have been generated, all or a portion of the permutations are processed on the dedicated Fantom services, until all permutations have been processed. The resulting rules are then forwarded to the final phase.

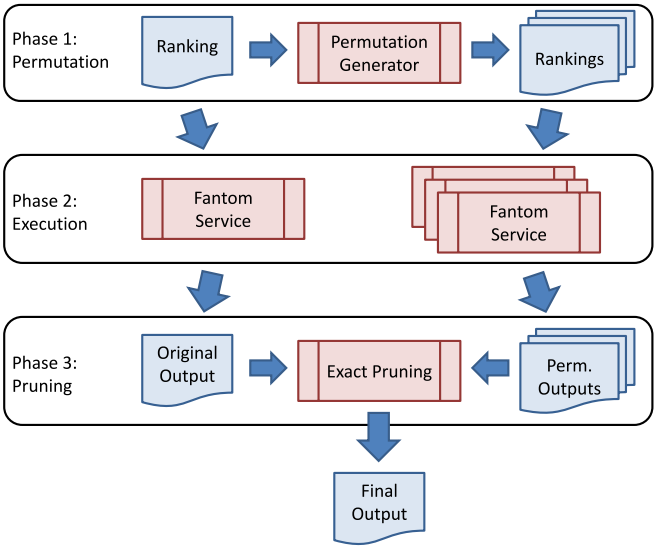


Fig. 1. Workflow for single-class exact optimization

In the third and final phase, the output is gathered and combined. First, the rules generated on the original input, from here on called the *original rule set*, are loaded into the system. After that, the rules generated on the permutations are loaded and compared to the original rule set. If there are any rules more specific than or equal to rules in the original rule set, and with a higher or equal score, then those rules in the original rule set are pruned.

4.2 Exact Testing: Multi-class Threshold Optimization

Fantom can also be used to perform multi-class comparison problems. The goal of these kinds of problems is to investigate what a specific group of the identifiers belonging to the *interest class* have in common, opposed to the rest of the identifiers belonging to the *control class*, which serve as contrast information to the interest class. When generating rules, subgroup rules that contain many identifiers in the interest class and few in the control class gain a higher score.

As an example of this kind of experiment, let us consider a wine comparison problem. Suppose we have a list of wines, all with unique identifiers, and we want to investigate what the 10 most popular wines have in common according to a certain wine ontology, for example the one described in the OWL documentation [26]. We would label the popular wines with $ClassInterest$, and the rest with $ClassControl$. When running Fantom, the service will strive to find all the rules that describe subgroups of the interest class, considering the control class purely as contrast information used in

the score measurement of the rules. Experiments of these kind are usually shorter and faster in Fantom, since the support threshold only applies to the interest class.

An interesting question here is: if we set specific thresholds on the interest group and control group, for what thresholds would we find the most interesting set of rules? To this end, a so-called *Permutation Counting Matrix* (PCM) was implemented.

A PCM is a structure that registers how many rules in the experiment apply to various threshold boundaries. On both axes, threshold boundaries are represented in percentages.

On the horizontal axis, also called the $max\_threshold$ axis, the control group is represented. Each threshold on this axes represents the percentage of identifiers in the entire control ranking that a rule can have at maximum. Note that if a rule satisfies a certain threshold x on a matrix of resolution $n \geq x$, then it satisfies all subsequent thresholds $x + 1, \ldots, n$.

On the vertical axis, which is called the $min\_threshold$ axis, the interest group is represented. In this case, each threshold represents the percentage of identifiers in the entire interest ranking that a rule can have at minimum. This implies that if a rule satisfies a certain threshold x on a matrix, then it satisfies all previous thresholds $1, \ldots, x$.

Consider PCM A below, which has a resolution of $m \cdot n$, whereby m represents the $min\_threshold$ resolution, and n is the $max\_threshold$ resolution.

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{pmatrix}$$

According to the properties discussed above, $a_{m,n} \geq a_{m,n-1} \geq \ldots \geq a_{m,1}$, and $a_{1,n} \geq a_{2,n} \geq \ldots \geq a_{m,n}$. This implies that the rules with potentially best scores have a high m index, while the n index is low, ultimately making $a_{m,1}$ a heaven point, containing rules that describe many interest group identifiers, while none or very little of the control group. However, these rules might not be the most interesting ones, if they are also generated in other experiments.

We can use the structure above to determine optimal threshold settings for interest and control classes, by using exact testing. With optimal, we mean threshold settings that have the most differential value for the original input with respect to the permutations. To calculate these settings, we again generate permutations on the original input. After that we generate the original PCM, and permutation PCMs for each of the permuted rankings. Finally, we subtract the permutation PCMs from the original one, and prune the rules of the original matrix simultaneously. The optimal thresholds are indicated by the matrix element with the highest value, and the ruleset returned are the rules adhering to those thresholds.

A workflow of multi-class exact testing is shown in Figure 2. As can be seen, phases one and two are the same, only phase three differs. Not only pruning takes place here, but also recombination of PCMs, and determination of the optimal class thresholds.

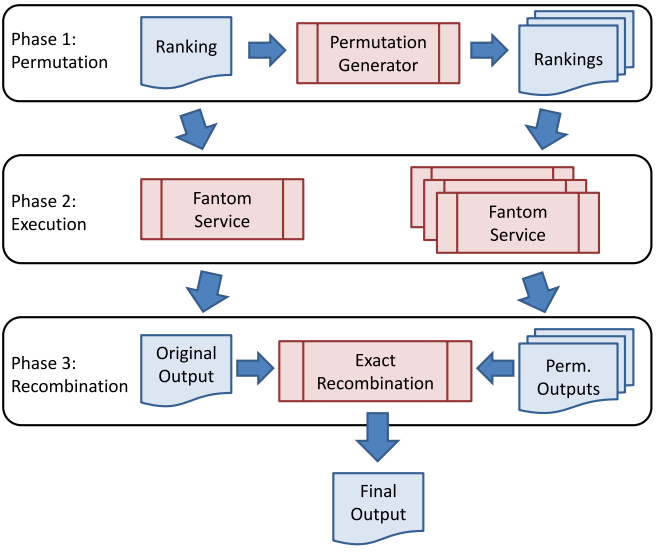


Fig. 2. Workflow for multi-class exact optimization

5 Experimental Results

In this section we discuss the experimental results of the two versions of exact testing with Fantom. We applied both versions on the AML vs. ALL publicly available microarray data set that compares gene expression profiles of AML and ALL [2], and present statistics on run times, pruning and parallelism.

Before we discuss the different experiments we will first provide a basic description of that data set. In the ALL vs. AML microarray data set, there are a total of $7,129$ probes, and 72 measurements per probe, 25 for AML patients, 47 for ALL. Mapping from probes to ENTREZ gene identifiers was performed with the Hu6800 annotations supplied by Affymetrix, and after elimination of probes that have no gene association or multiple ones, $5,445$ unique genes remained. To normalize the raw data, we used Quantile normalization again [7]. Rankings are obtained by performing t-value calculation with the Student's t-test between groups labelled with AML and ALL.

For the remainder of this section, all experiments were carried out on one or multiple machines all with the same configuration, namely an Intel Core Duo 2 times 2 GHz, with 4 GB of RAM.

5.1 Exact Testing: Single-Class Pruning

Single-class pruning is a process of three phases: permutation generation, rule discovery, and rule pruning. In the first phase of our experiments, permutations were generated from the microarray data set by swapping labels. By generating permutations in this way, instead of modifying values in the ranked list, we ensure that dependencies

between genes are preserved. Since there are 72 labels to consider, the total number of unique permutations would be:

$$\frac{72!}{25!\cdot 47!} = 1.53 * 10^{19}$$

Since this number exceeded our computational resources, which were limited to 16 computers in parallel, we generated 15 random permutations for each experiment, and ran all 16 simultaneously. We performed two different experiments, one generating rules that associates ontological concepts directly with genes, and another that uses the interaction option, indirectly associating genes with ontological terms through interaction with other genes. For each of these types, we performed three different experiments with low, medium, and high thresholds. To generate as many experiments as possible, we modified Fantom so that it will only allow at most one concept for each predicate per rule. Both the GO and KEGG ontologies were used which resulted in rules containing a maximum of four predicates.

Since interaction experiments are far more intensive in terms of computation and data access, a normal, unbiased way of experimentation resulted in a rule explosion that was unfeasible to be evaluated by experts. This is due to the interaction data, which documented many interactions, and thus many genes were associated to each ontological term. As a result, we had to insert a bias, declaring a list of genes that had to be in the rules. In our case, we declared this list to be the top 200 and bottom 200 genes, which are most differentially expressed. The participation threshold then only applies to the biased part instead of the entire ranking. This way, we avoided rule explosion at the cost of exhaustiveness. However, since the genes in the list were already at the top of the ranking, rules that could be found without a bias were likely to have a low interestingness. Note that due to the identifier association explosion, thresholds in the interaction experiments are still high in terms of participation.

First let us consider speed results of the experiments, shown in Table 1. We outlined several statistics for each configuration, such as participation threshold P , minimum interestingness score I, average completion time of the original input T_O in minutes (m) and seconds (s), lowest completion time of a permutation $min(T_{Pe})$, highest completion time of a permutation $max(T_{Pe})$, average completion time of permutations $avg(T_{Pe})$, and overhead of exact rule pruning T_{Pr}, which is the time of pruning the original rules by evaluating the outcome of the permutations. Note that for direct association experiments, all values are the averages of 20 runs, where in each run we

Table 1. Exact test pruning benchmarks

Profile	P	I	T_O	$min(T_{Pe})$	$max(T_{Pe})$	$avg(T_{Pe})$	T_{Pr}
Direct	8	0.40	2m	1m58s	2m	2m	5m41s
Direct	10	0.50	1m47s	1m41s	1m48s	1m44s	3m47s
Direct	12	0.70	1m23s	1m11s	1m26s	1m18s	18s
Interaction	60	0.35	233m05s	203m31s	222m18s	210m14s	17m41s
Interaction	75	0.40	149m51s	134m32s	148m22s	146m12s	10m11s
Interaction	80	0.50	129m50s	119m11s	131m27s	124m34s	1m27s

Table 2. Exact test pruning results

Profile	P	I	R_O	$min(R_{Pe})$	$max(R_{Pe})$	$avg(R_{Pe})$	$avg(R_{Pr})$
Direct	8	0.40	2,964	2,625	2,815	2,762	224
Direct	10	0.50	1,252	539	775	681	95
Direct	12	0.70	27	0	6	4	3
Interaction	60	0.35	26,212	17,544	21,996	20,112	1,766
Interaction	75	0.40	13,612	5,344	11,278	9,775	728
Interaction	80	0.50	4	0	2	1	0

generated different random permutations. Since interaction association experiments took much longer, all values of those experiments are the averages of 10 runs.

As can be seen in Table 1, interaction association experiments are much more data and processing intensive, yet exact pruning overhead increases much less dramatically. This is because options in interaction association experiments contain many genes associated to them, which makes the calculation of the maximum ES score very expensive. Exact pruning is only dependent on the number of rules in the original rule set and those of the permutations, which increase less dramatic.

Another observation is that execution times tend to increase exponentially with even a minor change in threshold setting, thus starting out with a high threshold and then moving to lower ones seems like the best strategy to apply.

A final observation is that rule generation times do not seem to differ much between the original input and the permutations, although experiments with permutations do seem to consistently take less time. This is because in some permutations very few rules could be generated with the given thresholds, sometimes even none, which reduces experiment times significantly.

Now let us consider pruning results of the experiments, shown in Table 2. We again outline several measurements for each configuration, such as the number of rules generated in the original input R_O, the minimum number of rules generated in the permutations $min(R_{Pe})$, the maximum number of rules generated in the permutations $max(R_{Pe})$, the average number of rules generated in the permutations $avg(R_{Pe})$, and the average number of rules pruned from the original set $avg(R_{Pr})$.

As expected, we can see in Table 2 that permutations structurally yield less rules, which explains the shorter experiment time. Pruning statistics do not differ much between direct and interaction association experiments, if we consider the last measurement an outlier. In both direct and interaction association experiments, higher settings result in relatively better pruning; at low settings around 6–7% of the original rules get pruned, while in higher settings this is as much as 11% (an interaction association experiment with a slightly less high setting resulted in pruning of about 10%).

5.2 Exact Testing: Multi-class Threshold Optimization

In multi-class experiments we investigate what the interest class or classes have in common with each other, or how they differ with respect to the rest of the identifiers that are

outside these classes. In our experiments we labelled all genes with a t-value of more than 0.5 as interesting, and compared this group against the rest of the genes in the input. Thus instead of the whole group of $5,445$ genes to be considered, the interest group now contained only 662 genes, while the rest was merely there for scoring purposes.

Participation thresholds in these experiments kept very low, since we want to uncover the optimal thresholds for which to return rules for to the user. Setting thresholds too high could interfere with this process, and yield a suboptimal result. Note that all participation thresholds set above now hold for the interest group instead of the whole ranking. This has implications for the permutations generated, since there have to be at least that many genes in the interest group to generate any rules at all. Therefore, a bit of bias in the permutations cannot be avoided, and thus we cannot speak of true randomness.

Once again we consider speed results of the experiments, which are shown in Table 3. All measured statistics and repetitive settings were kept the same as in Section 5.1.

Table 3. Exact test multi-class pruning benchmarks

Profile	P	I	T_O	$min(T_{Pe})$	$max(T_{Pe})$	$avg(T_{Pe})$	T_{Pr}
Direct	4	0.40	1m19s	1m12s	1m19sm	1m4s	8s
Direct	4	0.50	1m14s	1m8s	1m12s	1m11s	7s
Direct	4	0.70	1m7s	1m2s	1m10s	1m6s	5s
Interaction	30	0.35	610m05s	418m12s	584m12s	556m59s	44s
Interaction	30	0.50	608m33s	432m32s	600m17s	541m51s	32s
Interaction	30	0.60	605m41s	444m21s	567m22s	554m22s	11s

As can be seen in Table 3, compared to the single-class experiments, these experiments take less time for the direct associations, but more time for the interaction associations. This is due to the fact that direct association experiments are not negatively influenced as much by a lower threshold as interaction association experiments are. In interaction associations, groups in rules tend to experience an explosive growth, and this gets worse when lower thresholds are allowed.

Another observation is that experiment times seem to be rather constant. This is not surprising, since the experimental participation threshold was kept constant, and Fantom's pruning effect can really be seen in later stages of rule generation. Since we modified Fantom to only allow one ontological concept per predicate, those later stages never appear, and thus pruning on the basis of Enrichment Scoring is minimal.

Finally, it seems that pruning takes a very short time, indicating that there were few rules that could be used for pruning. This could either be because few rules were generated, or that the rules that were generated had a lower score than the original rules generated.

Now let us consider the pruning results, shown below in Table 4. Note that all descriptors are the same as in Table 2, with the addition of R_{Opt}, which is the resulting number of rules after pruning.

As can be seen, rule generation fluctuates more between the different thresholds than in a single-class experiment, but on the optimized thresholds the end result yields

Table 4. Exact test multi-class pruning results

Profile	P	I	R_O	R_{Opt}	$min(R_{Pe})$	$max(R_{Pe})$	$avg(R_{Pe})$	$avg(R_{Pr})$
Direct	4	0.40	1,182	638	0	5	3	2
Direct	4	0.50	946	524	0	3	2	0
Direct	4	0.70	436	227	0	1	1	0
Interaction	30	0.35	171,071	21,349	1,325	1,833	1,411	21
Interaction	30	0.50	11,113	4,453	0	121	32	8
Interaction	30	0.60	1	1	0	1	0	0

a pruning optimization in rules of 60 to 90 percent. The fact that initially more rules are generated is due to the lower thresholds.

We also see that again permutations do not yield many rules, and as a result additional exact pruning also has very little result. This is due to the design of the experiment. We set the threshold for over-expression to a t-value of 0.5. However, when generation permutations of the class labels, we influenced the t-values of the permutations, and thereby the ranking. There were less permutations that had sufficient genes with a t-value over 0.5, and when they did it was usually just barely over this threshold. Hence, permutations structurally yield less rules.

Another reason why pruning yields so little result is because of the low thresholds. Rules with small subgroups often contain very specific rules with high scores. In permutations, other genes are bound to be in the interest sets, yielding rules that are more specific in other parts of the ontology. As a result, these rules cannot be used to prune the original ones since they are not related to these rules.

6 Conclusions and Future Work

In this paper we discussed a subgroup discovery service called Fantom that finds subgroups given a set of weighed elements. We explained the technologies behind the algorithm, its data sources, and its way of combining that data to generate comprehensive patterns that are tailored to the expert knowledge of the researcher. We also showed how the service could be embedded in workflows to perform two ways of exact testing in order to prune Fantom outputs even further, and to optimize participation thresholds in case of a multi-class problem. We described how through a variation on Fisher's Exact Test we could prune more rules additional to the standard Fantom pruning algorithms, by generating permutations on the rankings. We also showed that with permutation testing we could find optimal participation thresholds for multi-class problems, as well as use the exact pruning method on it, but with less effect.

In single-class problems, both direct association and interaction association experiments had benefit from exact pruning, ranging from 6% with low thresholds to about 10% when thresholds are set high. A reason for this difference is that at high settings, there are not many rules, so if one does get pruned, the impact is much higher than on lower settings, where the impact is less. Therefore we predict that an overall performance of this algorithm is to prune about 6–8% of the rules.

In terms of performance, exact pruning on single-class problems is especially worthwhile in interaction association problems. Where in direct-association problems the overhead is sometimes grave in relative terms, e.g., more than 200%, it is relatively low for interaction association experiments, where the experiments themselves can sometimes take hours. Overall, overhead on interaction association experiments was between 1–7%, which was much better compared to the direct association experiments.

In multi-class problems, experimental time was more or less constant if the participation threshold was fixed, since pruning had minimal effect for rule generation for small collections of rule concepts. Since exact pruning had little effect, those pruning overheads were significantly lower for direct association experiments as well, between 7–10% of the experiment time. Overhead for the interaction association experiments was almost negligible.

Pruning due to optimized constraints of the rules yielded good results in both direct and interaction association experiments. For direct associations, pruning varied between 45–50% of the rules generated. For interaction associations, this number was higher, pruning from 65% to as much as 88% of the rules. Usually the optimal participation thresholds were greater than the minimum we applied, meaning that we could do these experiments faster with the same result. Exact pruning had almost no effect on both data sets, due to the experimental design and the fact that low thresholds create highly specific rules with high scores, which are unlikely to be pruned by rules generated from a permutation on the class labels, since that can modify the ranking profoundly.

Finally, we would like to discuss future work. Apart from pruning with exact tests, p-values for resulting rules could also be established with some modifications to the algorithm, although that would require more permutations than the 15 we generated. These p-values could give an even better indication on how special a specific rule is.

For multi-class problems, we kept the number of classes to two for now, but there are many problems that deal with more than two classes, so one interesting question is how to deal with those. Furthermore, we created a matrix based on participation thresholds. It would be interesting to see if the optimized rules would differ much when we optimize on the basis of score thresholds, or a mixture between score and participation.

References

1. Agrawal, R., Imielinski, T., Swami, A.N.: Mining association rules between sets of items in large databases. In: Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, pp. 207–216. ACM Press, New York (1993)
2. Armstrong, S.A., Staunton, J.E., Silverman, L.B., Pieters, R., den Boer, M.L., Minden, M.D., Sallan, S.E., Lander, E.S., Golub, T.R., Korsmeyer, S.J.: MLL translocations specify a distinct gene expression profile that distinguishes a unique leukemia. Nat. Genet. 30, 41–47 (2002)
3. Ashburner, M., Ball, C.A., Blake, J.A., Botstein, D., Butler, H., Cherry, J.M., Davis, A.P., Dolinski, K., Dwight, S.S., Eppig, J.T., Harris, M.A., Hill, D.P., Issel-Tarver, L., Kasarskis, A., Lewis, S., Matese, J.C., Richardson, J.E., Ringwald, M., Rubin, G.M., Sherlock, G.: Gene ontology: Tool for the unification of biology. The Gene Ontology Consortium. Nat. Genet. 25(1), 25–29 (2000)
4. Balaban, M.: The F-logic Approach for Description Languages. Annals of Mathematics and Artificial Intelligence 15, 19–60 (1995)

5. for Biotechnology Information, N.C.: GeneRIF – Gene Reference Into Functions (2009), http://www.ncbi.nlm.nih.gov/projects/GeneRIF/
6. Bodon, F.: A fast apriori implementation. In: FIMI 2003, Frequent Itemset Mining Implementations, Proceedings of the ICDM 2003 Workshop on Frequent Itemset Mining Implementations (2003)
7. Bolstad, B.M., Irizarry, R.A., Astrand, M., Speed, T.P.: A comparison of normalization methods for high density oligonucleotide array data based on variance and bias. Bioinformatics 19(2), 185–193 (2003)
8. Fisher, R.A.: On the interpretation of χ^2 from contingency tables, and the calculation of p. Journal of the Royal Statistical Society 85(1), 87–94 (1922)
9. Fisher, R.: Statistical methods for research workers, 13th edn. Biological monographs and manuals, vol. 5. Oliver and Boyd (1967)
10. Golub, T.R., Slonim, D.K., Tamayo, P., Huard, C., Gaasenbeek, M., Mesirov, J.P., Coller, H., Loh, M.L., Downing, J.R., Caligiuri, M.A., Bloomfield, C.D., Lander, E.S.: Molecular classification of cancer: Class discovery and class prediction by gene expression monitoring. Science 286(5439), 531–537 (1999)
11. Han, J., Pei, J., Yin, Y.: Mining frequent patterns without candidate generation. In: SIGMOD Conference, pp. 1–12. ACM, New York (2000)
12. Huang, D., Sherman, B., Tan, Q., Collins, J., Alvord, W.G., Roayaei, J., Stephens, R., Baseler, M., Lane, H.C., Lempicki, R.: The DAVID gene functional classification tool: A novel biological module-centric algorithm to functionally analyze large gene lists. Genome Biology 8(9), R183+ (2007)
13. Leung, E., Bushel, P.R.: PAGE: phase-shifted analysis of gene expression. Bioinformatics 22(3), 367–368 (2006)
14. Lu, Y., Rosenfeld, R., Simon, I., Nau, G.J., Bar-Joseph, Z.: A probabilistic generative model for go enrichment analysis. Nucl. Acids Res., 434+ (2008)
15. Maglott, D., Ostell, J., Pruitt, K.D., Tatusova, T.: Entrez gene: gene-centered information at ncbi. Nucleic Acids Res. 33(Database issue) (2005)
16. Mao, X., Cai, T., Olyarchuk, J.G.G., Wei, L.: Automated genome annotation and pathway identification using the kegg orthology (ko) as a controlled vocabulary. Bioinformatics 21(19), 3787–3793 (2005)
17. MyGrid: Taverna workbench 2.0 (2008), http://taverna.sourceforge.net/
18. Ogata, H., Goto, S., Sato, K., Fujibuchi, W., Bono, H., Kanehisa, M.: KEGG: Kyoto encyclopedia of genes and genomes. Nucleic Acids Res. 27(1), 29–34 (1999)
19. Online, C.D.: Ontology definition in information science (2007), http://www.computer-dictionary-online.org/ontology.htm?q=ontology
20. Pruitt, K.D., Tatusova, T., Maglott, D.R.: Ncbi reference sequences (refseq): A curated non-redundant sequence database of genomes, transcripts and proteins. Nucleic Acids Res. 35(Database issue), D61–D65 (2007)
21. Todorova, C., Stefanov, K.: Selection and use of domain ontologies in learning networks for lifelong competence development. In: Proceedings of the 2006 International Workshop on Learning Networks for Lifelong Competence Development, pp. 11–17. Springer, Heidelberg (2006)
22. Trajkovski, I., Lavrač, N., Tolar, J.: Segs: Search for enriched gene sets in microarray data. J. Biomed. Inform., 588–601 (2007)
23. Trajkovski, I., Zelezný, F., Tolar, J., Lavrac, N.: Relational subgroup discovery for descriptive analysis of microarray data. In: R. Berthold, M., Glen, R.C., Fischer, I. (eds.) CompLife 2006. LNCS (LNBI), vol. 4216, pp. 86–96. Springer, Heidelberg (2006)

24. Vastrik, I., D'Eustachio, P., Schmidt, E., Joshi-Tope, G., Gopinath, G., Croft, D., de Bono, B., Gillespie, M., Jassal, B., Lewis, S., Matthews, L., Wu, G., Birney, E., Stein, L.: Reactome: A knowledgebase of biological pathways and processes. Genome Biology 8, 39+ (2007)
25. W3C, T.W.W.W.C.: Resource description framework, rdf (2004), http://www.w3.org/RDF/
26. W3C, T.W.W.W.C.: Web ontology language, OWL (2004), http://www.w3.org/2004/OWL/
27. Wain, H.M., Lush, M., Ducluzeau, F., Povey, S.: Genew: The human gene nomenclature database. Nucleic Acids Research 30(3), 169–171 (2002)
28. Zheng, Q., Wang, X.J.J.: GOEAST: A web-based software toolkit for gene ontology enrichment analysis. Nucleic Acids Research, 358–363 (2008)

Semantically-Guided Workflow Construction in Taverna: The SADI and BioMoby Plug-Ins

David Withers, Edward Kawas, Luke McCarthy, Benjamin Vandervalk, and Mark Wilkinson

Heart + Lung Institute at St. Paul's Hospital,
University of British Columbia, Vancouver, BC, Canada
markw@illuminae.com

Abstract. In the Taverna workflow design and enactment tool, users often find it difficult to both manually discover a service or workflow fragment that executes a desired operation on a piece of data (both semantically and syntactically), and correctly connect that service into the workflow such that appropriate connections are made between input and output data elements. The BioMoby project, and its successor the SADI project, embed semantics into their data-structures in an attempt to make the purpose and functionality of a Web Service more computable, and thereby facilitate service discovery during workflow construction. In this article, we compare and contrast the functionality of the BioMoby and SADI plug-ins to Taverna, with a particular focus on how they attempt to simplify workflow synthesis by end-users. We then compare these functionalities with other workflow-like clients we (and others) have created for the BioMoby and SADI systems, discuss the limitations to manual workflow synthesis, and contrast these with the opportunities we have found for fully automated workflow synthesis using the semantics of SADI.

Keywords: Semantic Web, Web Services, SADI, BioMoby, Taverna, workflow.

1 Introduction

Biology is increasingly becoming an *in silico* science, largely as a result of the rapid advance of high-throughput technologies for DNA sequencing, protein analysis, gene expression, metabolic profiling, and genotyping. Many studies currently being undertaken in bio/medicine require access to, integration of, and analysis of some or all of these data types. The scale, scope, and complexity of these analyses makes the traditional approach of copy/pasting data into Web pages untenable, and thus has led to the emergence of Workflows as a primary "object" in modern biology [1]. This shift from manual Web-based analysis to semi- or fully-automated analytical pipelines has been mirrored by the concomitant emergence of software and community support for this new *in silico* paradigm.

T. Margaria and B. Steffen (Eds.): ISoLA 2010, Part I, LNCS 6415, pp. 301–312, 2010.

For non-coders, workflow design and enactment software makes it possible to conduct the kinds of high-throughput analyses that were formerly the exclusive domain of bioinformatics professionals. Moreover, emergent public workflow repositories like myExperiment [2] will increasingly play a role in supporting workflow construction by end-users - either through reuse, extension, or re-purposing of other researchers workflows [3]. Nevertheless, recent studies [4, 5, 6] have demonstrated that end-user biologists continue to have problems constructing useful or functional workflows, even when presented with existing scaffolds or templates. This is, at least in part, due to the difficulty of manually discovering a service or workflow fragment that does what is necessary, and correctly integrating that service into the workflow, both syntactically and semantically.

Taverna is a general-purpose workflow design tool designed to manage most "flavours" of Web Service (CGI, SOAP, BioMoby, etc.), and handle data flow related to any domain of investigation [7]. This broad support is important given that many biomedical investigations require access to various types of biological and chemical data, pathway data, and statistical algorithms; however it comes at a cost of complexity – that is, the number of Web Services available in the default Taverna interface is already on the order of 3000 or more, and adding additional service endpoints is straightforward. This makes Taverna distinct from some other workflow tools in the bioinformatics space (for example, GenePattern [8]), where the available services are pre-screened and pre-selected by the software designers to solve specific problems, and therefore only number in the tens or hundreds of services. Consequently, however, Taverna suffers an embarrassment of riches, or more precisely, the end-users of Taverna "suffer" when they must identify and chose the service they require from a vast myriad of options at any given point in workflow construction.

One of the earliest examples of semantically-supported Web Service discovery was exhibited by the TAMBIS (Transparent Access to Multiple Bioinformatics Information Sources) project [9]. Since then, a wide variety of initiatives have taken similar approaches to improving service discovery (three early high-profile projects are reviewed in Lord et al. [10]). The common thread between all of them is that their individual service registries contain semantic information beyond that provided by the Service's WSDL file. Here we are going to compare and contrast only two of these – BioMoby [11], and the SADI Framework [12] – since both of these Semantic Web Service frameworks have created plug-ins to Taverna that leverage the additional semantic search power of their respective registries.

In the remainder of this article we will first briefly describe the BioMoby and SADI Semantic Web Service frameworks. We will then describe the functionality of their respective plug-ins to Taverna, and discuss how the nature of their support for Web Service discovery differs. Finally, we compare these functionalities with other workflow-like clients we (and others) have created for the BioMoby and SADI systems, discuss the limitations to manual workflow synthesis, and contrast these with the opportunities we have found for fully automated workflow synthesis using the semantics of SADI.

2 BioMoby Semantic Web Services

BioMoby was initiated in 2001 from within the model organism database community. It aimed to standardize methodologies to facilitate interoperable information exchange

and access to analytical resources by creating a community-built and community-curated ontology of bioinformatics data-types and analytical operations.

The key to BioMoby's interoperable behaviours was its invention of a 'boutique' syntax in the form of an ontology-based XML schema – an instance of any ontological node had a specific and predictable XML representation that could be automatically determined by traversing the ontology. Thus machines could receive information of an unknown type, and determine not only its XML structure, but also the "meaning" of every sub-structure, by referring to the BioMoby data-type ontology. All BioMoby-compliant Web Services consumed and produced data in this boutique XML schema, thus services consuming ontologically-compatible data-types were, by definition, interoperable (at least syntactically, and to a large degree semantically).

Service discovery in Moby is accomplished through querying a centralized registry ("Moby Central") where Services are indexed by ontology-based input data-type, ontology-based output data-type, a controlled vocabulary of service functionality types, and by service provider identification. It is important to emphasize that Moby Central, like traditional Web Services registries, indexes the input and output data-types as "globs" of data - effectively, a reference to an XML schema - and therefore does not explicitly expose the finer sub-structure of the data[1]. This observation is critical to understanding the contrast between BioMoby and the SADI framework described in section 3.

Despite being key to interoperability, the BioMoby data-type ontology was its primary weakness. Not only was the XML representation of the ontology project-specific, the ontology itself took the form of a large centralized resource that, while being openly community-editable, still required community agreement and consensus buy-in to ensure interoperability. In practice, this buy-in was only tentative, and many providers created duplicate data-types with different names and in some cases duplicated entire sub-branches of the ontology to be specific to their needs/terminologies/projects. Thus, as new standards for representing and publishing ontologies became available from the W3C, we undertook to invent a new semantic web service framework – SADI (Semantic Automated Discovery and Integration) – guided by the successes and failures of the BioMoby project.

3 SADI Semantic Web Services

SADI is a set of guidelines for Semantic Web Service provision that aim to maximize interoperability between Web Services while minimizing the complexity of service provision by the resource providers. While SADI does not invent any new technologies or standards, project-relevant codebases in Java and Perl have been developed to support Web Services compliant with these guidelines and best-practices.

Like the BioMoby project, SADI embeds semantics into its data-structures in an attempt to make the purpose and functionality of a Web Service more computable. Where BioMoby used a boutique XML serialization of an ontology in order to represent semantically-grounded messages, SADI utilizes the W3C standards of Resource

[1] In principle, such sub-structures could be determined by exploring the data-type ontology; however in practice no client application ever did this.

Description Framework (RDF) and Web Ontology Language (OWL) and their respective XML serializations. Service input and output are defined as OWL-DL classes, and OWL Individuals of these classes (in RDF) are consumed and produced during service invocation.

The key novelty of the SADI project lies in one very simple best-practice guideline – that is, that the identity of the input RDF node, and the identity of the output RDF node, must be identical. As a consequence of this, every service becomes an annotation service, where the input node is "decorated" with labeled RDF relationships to new data nodes derived from the execution of the service. As an extension to this, since input and output Classes are defined in OWL-DL (where property and value restrictions are used to describe the features of input and output class membership), it therefore becomes possible to determine what features are added to an input node simply by examining the difference between the input and output OWL Class definitions, since these are, by definition, the same "entity" (URI). These features are indexed in a registry and used for Service discovery. Note that this is subtly, but critically, different from the BioMoby registry index – where BioMoby indexes only the data-type "glob", in SADI, the properties added to the input by the service are indexed in the registry, since these are part of the output data-type definition. Thus, in SADI, we can support much finer-grained searches of the data that is output from participating services.

Service discovery in SADI is based on searches for services that consume a particular set of data properties, and produce one or more new properties of-interest based on those properties. Note again that searching is rarely, if ever, done for a particular output based on its *Class,* but rather for sets of specific data properties in relation to the input data. This approach was designed to mimic the linguistics of scientific query, where researchers frequently ask questions about the relationship between two pieces of data (e.g. "what is the coding sequence ***of*** the BRCA1 gene?"). While in BioMoby one could search for a service that consumed the data-type "gene name" and produced the data-type "nucleotide sequence", there would be ambiguity about what the exact relationship between that gene and that sequence was (e.g. is it the gene sequence, the coding sequence, the promoter sequence, or a contig that contained that gene, etc.). Conversely, in SADI, one might search for services that provide the "hasCodingSequence" property on gene names, thus leaving little ambiguity about what the service does.

4 The Taverna BioMoby and SADI Plugins

BioMoby and SADI are both products, in part, of the Genome Canada Bioinformatics Platform, where Web Service Workflows are the means by which we provide bioinformatics support to Platform end-users. The Platform has chosen the Taverna client application as one of its primary end-user tools due to its powerful, yet straightforward interface. In total, >1500 BioMoby Services, and an increasing number of SADI services (currently near 100) are available. However, as discussed earlier, the large number of Web Services available in the Taverna interface makes workflow construction a challenge for even expert end-users. As such, we have undertaken to create plug-ins to Taverna that make it easier for our biologist end-users to work with BioMoby and SADI Services, both at the level of Service discovery and the level of correct "wiring" of Services into workflows.

4.1 The BioMoby Plugin to Taverna

The BioMoby plugin is described in detail elsewhere [13], but we will recap the salient features here. When searching for a Service there are two common scenarios: either the user has a particular data-type that they want to submit to a Service for analysis, or the user has already created a fragment of a workflow and now wish to pipeline the output from that workflow into a new Service. In BioMoby, both of these cases are identical in that both a standalone piece of data, and a Service output, are strictly typed by the BioMoby data-type ontology. Thus the registry query executed by the Taverna plugin is for services that consume data-type "X" as their input. This search can be enhanced by asking the registry to search for services that consume "X", or an ontological parent-type of "X", as their input (or conversely, which services produce "X" as their output when constructing workflows in the reverse direction) as shown in Figure 1. The resulting matches can be ordered by service name (a human-readable and often semi-informative string), by service type (a controlled vocabulary describing kinds of bioinformatics operations), or by service provider (by their unique provider URI string) as shown in Figure 2. In addition, a discovered service can be further examined to determine what BioMoby data-type it will output if invoked.

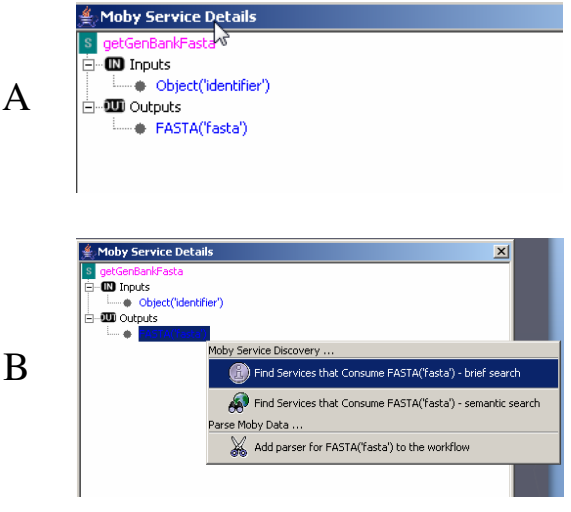


Fig. 1. The BioMoby Web Service search interface. As shown in A, a Web Service (getGenBankFASTA) can be examined to show the BioMoby datatypes consumed and produced by its input and output ports. In this case, the input port consumes the BioMoby data-type "Object" (which is used to pass database identifiers) and its output port produces the BioMoby datatype "FASTA". The words in brackets following the data-type names are human readable annotations of the data-type added by the service provider to help explain the purpose of each input and output parameter. This data-type information can be used, by right-clicking on the data-type, to discover BioMoby services capable of consuming (as shown in B) or producing, that datatype. This allows workflow construction to be achieved either in the forward or reverse direction.

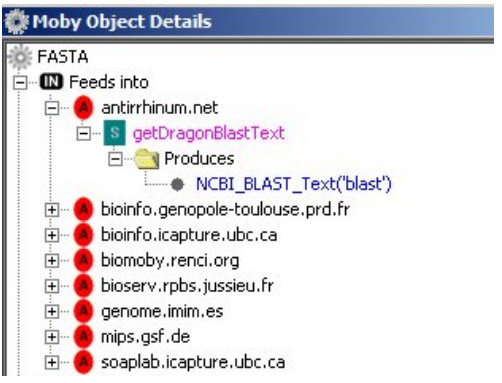


Fig. 2. The results of searching for services that consume FASTA (see figure 1B). In this view, the results are sorted by service provider; the provider "antirrhinum.net" is expanded to reveal the service – getDragonBlastText – that matches those criterion. In addition, the data-type output from that service, NCBI_Blast_Text, is provided in the expanded view to assist the user in determining if this service is likely to be appropriate for their needs.

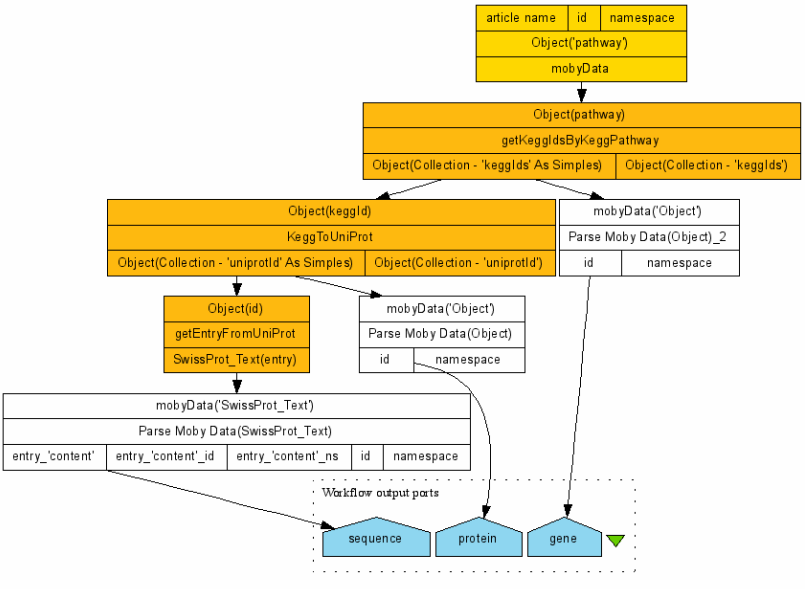


Fig. 3. A BioMoby workflow that extracts the gene names, protein names, and protein sequences that participate in a given biochemical pathway in the KEGG database. Orange nodes are BioMoby services; white nodes are parsers specific to BioMoby data types to enable extraction of data from BioMoby's "boutique" XML schema; blue pentagons are output data buckets. In each node, the top layer describes the input 'ports' by their data-type name (e.g. "Object" – a BioMoby datatype for record identifiers), the middle layer is the service name, and the lower layer describes the output 'ports' by their data type name (e.g. "SwissProt_Text"). For both input and output ports, the human readable parameter name of the port is in brackets (e.g. "keggId").

When the desired Service is selected, the Taverna plugin automatically connects it to the workflow; this connection is guaranteed to be correct since the strict data- typing of BioMoby, and the ontological regulation of its data structures, ensures that data can be passed verbatim from one service to another so long as the data-types are ontologically related.

A typical workflow resulting from this iterative discover/connect process is shown in Figure 3. Notice that the widgets representing each service show the transformative nature of that service – i.e., the data-type that goes in, and the data-type that comes out, are displayed on the widget. The annotations of those data-types - effectively, a human-readable single-word name given to the input or output – are also displayed, but may or may not be informative.

4.2 The SADI Plugin to Taverna

Superficially, the SADI plugin to Taverna has a near-identical functionality to that of the BioMoby plugin. Generally speaking, the starting point of service discovery is the same – either a specific data-type, or an existing workflow fragment. The search against the SADI registry is, therefore, for services that consume a particular type of data, as shown in Figure 4. What differs from BioMoby, however, is how that data-type is described. In a SADI search, the information sent to the registry query is an OWL Class-name, rather than a (relatively) opaque BioMoby data-type name.

As such, the search API that accesses the SADI registry can use the property restrictions within that OWL Class definition to search for SADI services that consume sub-features of the output data. Thus, the SADI search is more semantically rich than a BioMoby search, since arbitrary sub-sets of data properties will discover services capable of consuming those subsets, rather than strictly matching based on a hierarchical data-type ontology.

This subtle distinction is perhaps best described with an example. Some service Foo attaches the properties of "protein identifier", "molecular weight", and "Gene Ontology annotation" to its input. Foo can then be used to discover a downstream service Bar that consumes only "molecular weight" as its input, or combinations of

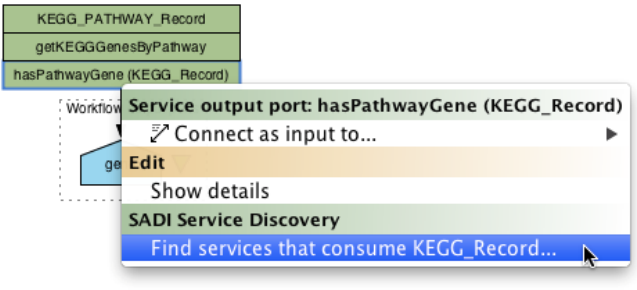


Fig. 4. The search interface of the SADI plugin. Search can now be conducted by right-clicking on the service widget itself in the workflow display, which is likely to be more intuitive for the biologist end-user who expects that workflow diagram to be interactive [4]. Search is simply a matter of clicking the "Find services" option of the pop-up menu.

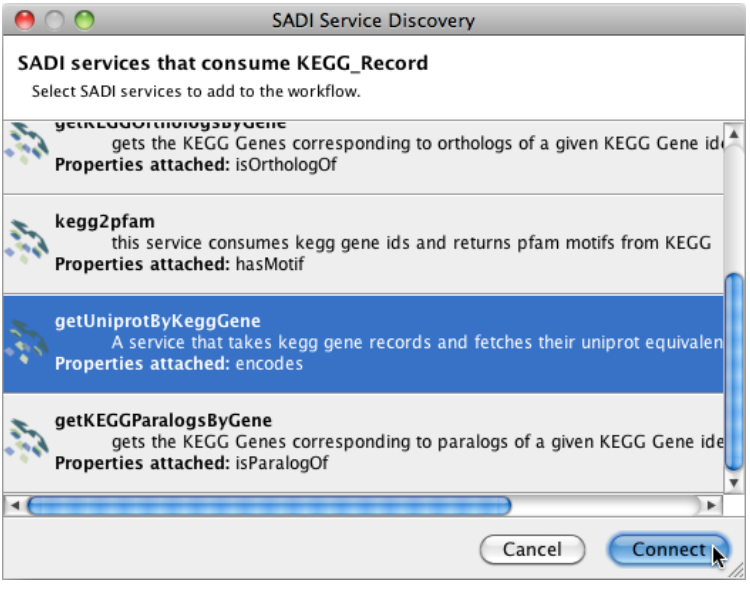


Fig. 5. The results of searching for services that consume KEGG_Record (see figure 4). All valid services are displayed, together with some metadata about the service (service name, and human-readable description of service functionality), and the properties that the service will attach to its input data. Note the subtle distinction that, unlike the BioMoby plug-in, it is the name of the property, rather than its data-type, which is displayed to the end-user.

"protein identifiers" and "Gene Ontology annotations" as its input. The matching is accomplished by a DL reasoner, which compares the properties in-hand to the input OWL property restrictions of each service in the registry. Once discovered, Foo and Bar can be connected accurately, both syntactically and semantically, without human intervention.

A SADI workflow is shown in Figure 6 that is functionally identical to the BioMoby workflow shown in Figure 3. The salient features to note in this diagram are: (1) The output ports describe the properties being generated by the service, and the data-types of those properties. This makes the service functionality extremely transparent, for example, the upper green box would read "A KEGG pathway has a pathway gene that is represented as a KEGG Record"; (2) There is visual confirmation to the end-user that their workflow is "correct" because the naming of record identifier data-types in OWL is typically more human-readable than the equivalent BioMoby data-type ontology term. For example, the KEGG_Record OWL class would have simply been "Object" in the BioMoby data-typing ontology (with an additional attribute "namespace=KEGG_Record"; however this attribute/value pair is not amenable to logical reasoning in any strict sense). Thus by visual inspection the user can see that the KEGG_Record from the first service is being fed as input to the KEGG_Record slot of the subsequent service. We have not yet tested the effectiveness of these small interface changes on end-user utility; however we suspect that we will see an improvement.

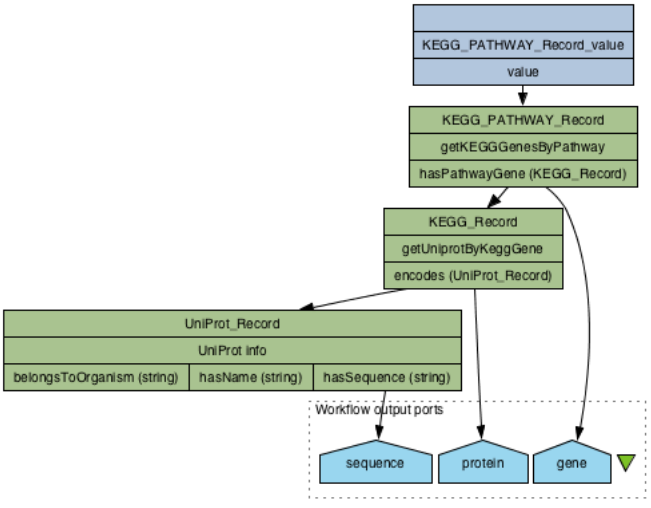


Fig. 6. A SADI workflow that extracts the gene names, protein names, and protein sequences that participate in a given biochemical pathway in the KEGG database. Square blue nodes are data inputs, green nodes are SADI Services; blue pentagons are data buckets. For each SADI service, the top layer is the input data-type (the OWL Class Name), the middle layer is the service name, and the bottom layer is the list of properties and value restrictions provided by that service. For example, the getUniProtByKeggGene Service consumes individuals of class KEGG_Record and attaches the "encodes" property which will have a value that is of type UniProt_Record.

5 Semantic Service Discovery in Workflow Construction

Most workflow composition systems support some form of assisted service discovery. Kepler allows the use of ontologies to describe the input and output of workflow components [14], and similarly Galaxy [15] supports both semantic service matching (using Lumina [16]), as well as ontologically-grounded lifting and lowering XML Schema, in a manner that mirrors the SAWSDL standard from the W3C [17]. The semantics of the service operation are also searchable in these systems through OWL-S/WSDL-S-like standards, where the pre-state, post-state, and functional operations of a service are semantically described.

BioMoby and SADI took more simplistic, but mutually distinct approaches to describing the semantics of a Web Service. In BioMoby, the semantics of the input and output messages are embedded in the message itself through its boutique XML serialization of the BioMoby data type ontology. The semantics of a BioMoby service operation are described in a simplistic hierarchy of bioinformatics service types akin to the ontological hierarchy created by the myGrid project [18] for their FETA Semantic Web Service annotation system [19]. In SADI the semantics of the input and output messages, and the semantics of the service operation itself, are *both* embedded in the message. This is because the SADI Semantic Web Service framework requires that service input and output RDF graphs have the same subject URI. Thus, the OWL

classes that describe the input and output data-types also, implicitly, describe the difference between those data-types. This becomes an important descriptor of the functionality of the service during search, since it explicitly describes how the input and output data relate to one another. SADI does not rely on a centralized ontology to define these relationships, but rather allows the service provider to chose, or publish, *any* ontological predicate that semantically describes this relationship. The embedding of semantics directly in the message in both cases has significant consequences on how services are discovered.

For the BioMoby and SADI Taverna plug-ins, semantically-guided discovery of services differs in two primary ways:

1. As just described, SADI searches lack a distinct searchable feature regarding service functionality. While nothing about the SADI framework precludes the addition of detailed annotations of SADI service operations, we have not yet found a need for this; the semantic relationship between input and output is highly descriptive of what the functionality of the service *must be*, and has been sufficient to resolve our use-cases to date, including that of the Taverna plug-in. Moreover, while it is unlikely that this simplistic annotation will ever support fully automated workflow synthesis, there is evidence that such automation is not needed, or even wanted, by our target end-users [10]. Thus the simplification of use we gain is not, so far, detrimentally offset by a lack of support for full automation. Further, several studies [4; 6] have shown that, for our target end-users, the concept of "data-typing" is extremely foreign (described as "incomprehensible" by Gordon and Sensen) and we are hopeful that indexing services based on data-type relationships, rather than (or in addition to) data-types, will facilitate their use of the discovery tools.
2. Second, service discovery with the SADI plug-in allows richer semantic matching because DL reasoners can match subsets of properties within complex data objects with services capable of consuming those sub-properties. Planned user-studies which mirror those of Gordon and Sensen will determine if this additional semantic discovery-power simplifies, or complicates, the process of rational workflow construction by our target audience.

6 Other BioMoby/SADI Web Service Composition Systems

Limiting the discussion only to BioMoby and SADI Semantic Web Services, a wide array of client applications support semi- or fully-automated workflow synthesis with one or both of these frameworks. In addition to Taverna, BioMoby workflow clients include other standalone clients such as Seahawk [20], jORCA [21], and Remora [22]; web-based clients such as Gbrowse Moby[23], MOWserv [24], and Mobyle [25]; "aggregators" such as Jabba [26] and DataBiNS [27] where pre-determined Service workflows are called to create Web-page content; and finally automated workflow construction systems such as Magellenes [28], where the user specifies their starting and ending data-type, and the system determines paths through the analytical service-space that will derive that output from that input. The newer SADI Semantic Web Service system has fewer clients so far, but both web based and standalone tools are already available. These include SHARE [29] and the Taverna plug-in described here.

Of particular interest in the context of this manuscript is the SHARE client, because of its novel approach to automated workflow synthesis. SHARE automates workflow composition using the property constraints within OWL-DL Class definitions as a guide. While in Magellenes the user is presented with a selection of reasonable pre-constructed workflows, SHARE can (often) precisely determine the path by which a complex data-type can be synthesized simply by examining the property and property-value constraints within the data-type's definition. This is possible specifically because the SADI framework, unlike BioMoby, utilizes both data-type and relationship information in its service annotation.

While the required usability studies on the new Taverna plug-in have not yet been undertaken, we have observed notable added service-discovery power in other SADI client applications using a similar property-based searching paradigm. Moreover, there is mounting evidence that our target end-users find it difficult to comprehend and work with data-types. As such, we feel confident that these same end-users will find this new paradigm of working with the properties of data, rather than with explicit data-types to be a much more natural way of approaching workflow construction.

Acknowledgments. The BioMoby project was funded in part by Genome Canada and Genome Prairie through the Genome Canada Bioinformatics Platform. The SADI project was funded by the Heart and Stroke Foundation of BC and Yukon, Microsoft Research, and the CIHR. Development of the BioMoby and SADI plugins to Taverna have been funded in part by Genome Canada and Genome Prairie, by expertise donated from the myGrid project, and by CANARIE through its funding of the C-BRASS Project. Core laboratory funding is derived from an award from NSERC.

References

1. Goderis, A., Li, P., Goble, C.: Workflow Discovery: Requirements from E-science and a Graph-based Solution. International Journal of Web Services Research 5(4) (2008)
2. Goble, C., DeRoure, D.C.: myExperiment: social networking for workflow-using e-scientists. In: Proceedings of the 2nd Workshop on Workflows in Support of Large-Scale Science, pp. 1–2 (2007)
3. Wroe, C., Goble, C., Goderis, A., Lord, P., Miles, S., Papay, J., Alper, P., Moreau, L.: Recycling workflows and services through discovery and reuse. Concurrency Computat: Pract. Exper. 19(2), 1–7 (2006)
4. Gordon, P.M.K., Sensen, C.: A Pilot Study into the Usability of a Scientific Workflow Construction Tool. Technical Report #2007-874-26. Department of Computer Science, The University of Calgary (2007)
5. Goderis, A., Sattler, U., Lord, P., Goble, C.: Seven Bottlenecks to Workflow Reuse and Repurposing. In: Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A. (eds.) ISWC 2005. LNCS, vol. 3729, pp. 323–337. Springer, Heidelberg (2005)
6. Gordon, P.M.K., Barker, K., Sensen, C.W.: Helping Molecular Biologists Effectively Build Workflows, Without Programming. In: Lambrix, P., Kemp, G. (eds.) Proceedings of 7th International Conference on Data Integration in the Life Sciences (DILS 2010), Gothenburg, Sweden, August 25-27, pp . 74–89 (2010)
7. Oinn, T., Greenwood, M., Addis, M., Alpdemir, N., Ferris, J., Glover, K., Goble, C., Goderis, A., Hull, D., Marvin, D., Li, P., Lord, P., Pocock, M., Senger, M., Stevens, R., Wipat, A., Wroe, C.: Taverna: lessons in creating a workflow environment for the life sciences. Concurrency Computat: Pract. Exper. 18(10), 1067–1100 (2006)

8. Reichm, M., Liefeld, T., Gould, J., Lerner, J., Tamayo, P., Miserov, J.P.: GenePattern 2.0. Nat. Genet. 38(5), 500–501 (2006)
9. Stevens, R., Baker, P., Bechhofer, S., Ng, G., Jacoby, A., Paton, N.W., Goble, C.A., Brass, A.: Tambis: transparent access to multiple bioinformatics information sources. Bioinformatics 16(2), 184–185 (2000)
10. Lord, P., Bechhofer, S., Wilkinson, M.D., Schiltz, G., Gessler, D., Hull, D., Goble, C., Stein, L.: Applying semantic web services to Bioinformatics: Experiences gained, lessons learnt. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) ISWC 2004. LNCS, vol. 3298, pp. 350–364. Springer, Heidelberg (2004)
11. The BioMoby Consortium. Interoperability with Moby 1.0 - It's better than sharing your toothbrush! Briefings in Bioinformatics 9(3), 220–231 (2008)
12. Wilkinson, M.D. Vandervalk, B. McCarthy, L.: SADI Semantic Web Services - cause you can't always GET what you want! In: IEEE Asia-Pacific Services Computing Conference, APSCC 2009, pp. 13–18 (2009)
13. Kawas, E., Senger, M., Wilkinson, M.D.: BioMoby extensions to the Taverna workflow management and enactment software. BMC Bioinformatics 7(523) (2006)
14. Pignotti, E., Edwards, P., Preece, A., Gotts, N., Polhill, G.: Enhancing Workflow with a Semantic Description of Scientific Intent. In: Bechhofer, S., Hauswirth, M., Hoffmann, J., Koubarakis, M. (eds.) ESWC 2008. LNCS, vol. 5021, pp. 644–658. Springer, Heidelberg (2008)
15. Taylor, J., Schenck, I., Blankenberg, D., Nekrutenko, A.: Using galaxy to perform large-scale interactive data analyses. Curr. Protoc. Bioinformatics, ch. 10:Unit 10.5 (September 2007)
16. `http://lsdis.cs.uga.edu/projects/meteor-s/downloads/Lumina/files/thesis.pdf` (Downloaded May 15, 2010)
17. `http://www.w3.org/2002/ws/sawsdl/` (Downloaded May 15, 2010)
18. Wolstencroft, K., Alper, P., Hull, D., Wroe, C., Lord, P.W., Stevens, R.D., Goble, C.A.: The myGrid ontology: bioinformatics service discovery. International Journal of Bioinformatics Research and Applications 3(3), 303–325 (2007)
19. Lord, P., Alper, P., Wroe, C., Goble, C.: Feta: A light-weight architecture for user oriented semantic service discovery. In: Gómez-Pérez, A., Euzenat, J. (eds.) ESWC 2005. LNCS, vol. 3532, pp. 17–31. Springer, Heidelberg (2005)
20. Gordon, P.M.K., Sensen, C.: Seahawk: moving beyond HTML in Web-based bioinformatics analysis. BMC Bioinformatics 8(208) (2007)
21. Martín-Requena, V., Ríos, J., García, M., Ramírez, S., Trelles, O.: jORCA: easily integrating bioinformatics Web Services. Bioinformatics 26(4), 553–559 (2010)
22. Carrere, S., Gouzy, J.: REMORA: a pilot in the ocean of BioMoby web-services. Bioinformatics 22(7), 900–901 (2006)
23. Wilkinson, M.: Gbrowse Moby: A Web-based browser for BioMOBY Services. Source Code for Biology and Medicine 1(4) (2006)
24. Navas, I., Rojano, M., Ramirez, S., Pérez, A.J., Aldana, J.F., Trelles, O.: Intelligent client for integrating bioinformatics services. Bioinformatics 22, 106–111 (2006)
25. Néron, B., Ménager, H., Maufrais, C., Joly, N., Maupetit, J., Letort, S., Carrere, S., Tuffery, P., Letondal, C.: Mobyle: a new full web bioinformatics framework. Bioinformatics 25(22), 3005–3011 (2006)
26. `http://bioinfo.mpiz-koeln.mpg.de/jabba/help.html` (Downloaded May 15, 2010)
27. Song, Y.C., Kawas, E., Good, B.M., Wilkinson, M.D., Tebbutt, S.: DataBiNS: a BioMoby-based data-mining workflow for biological pathways and non-synonymous SNPs. Bioinformatics 23(6), 780–782 (2007)
28. Ríos, J., Karlsson, J., Trelles, O.: Magallanes: a web services discovery and automatic workflow composition tool. BMC Bioinformatics 10(334) (2009)
29. Vandervalk, B.P., McCarthy, E.L., Wilkinson, M.D.: SHARE: A Semantic Web Query Engine for Bioinformatics. In: Gómez-Pérez, A., Yu, Y., Ding, Y. (eds.) ASWC 2009. LNCS, vol. 5926, pp. 367–369. Springer, Heidelberg (2009)

Workflow Construction for Service-Oriented Knowledge Discovery

Vid Podpečan[1], Monika Žakova[2], and Nada Lavrač[1,3]

[1] Jožef Stefan Institute, Ljubljana, Slovenia
[2] Czech Technical University, Prague, Czech Republic
[3] University of Nova Gorica, Nova Gorica, Slovenia

Abstract. The paper proposes a Service-oriented Knowledge Discovery (SoKD) framework and a prototype implementation named Orange4WS. To provide the proposed framework with semantics, we are using the Knowledge Discovery Ontology which defines relationships among the ingredients of knowledge discovery scenarios. It enables to reason which algorithms can be used to produce the results required by a specified knowledge discovery task, and to query the results of the knowledge discovery tasks. In addition, the ontology can also be used for automatic annotation of manually created workflows facilitating their reuse. Thus, the proposed framework provides an approach to third generation data mining: integration of distributed, heterogeneous data and knowledge resources and software into a coherent and effective knowledge discovery process. The abilities of the prototype implementation have been demonstrated on a text mining use case featuring publicly available data repositories, specialized algorithms, and third-party data analysis tools.

1 Introduction

Fast-growing volumes of complex and geographically dispersed information and knowledge sources publicly available on the web present huge opportunities (and challenges) for knowledge discovery systems. A principled fusion and mining of such relevant data requires the interplay of diverse specialized algorithms resulting in highly intricate workflows.

If such workflows are built on top of a service-oriented architecture, the processing of workflow ingredients (e.g. data mining algorithms) can be distributed between the user's computer and a remote computer system. Moreover, if data mining algorithms are implemented as web services, they are self-contained, platform independent, and do not need the entire environment (e.g., Weka). Obviously, this greatly expands the range of potential users as, e.g., Weka algorithms could - via a service-oriented approach - become integrated into any software solution capable of making use of web services. Finally, if the services are maintained and updated regularly, all of their users will always have access to the latest implementations without any effort.

While the mutual relations of specialized algorithms used in the workflows and principles of their applicability may be mastered by computer scientists, their

T. Margaria and B. Steffen (Eds.): ISoLA 2010, Part I, LNCS 6415, pp. 313–327, 2010.

command cannot be expected from the end user, e.g. a life scientist. A formal capture of this knowledge is thus needed, e.g. in the form of ontologies of relevant services and knowledge/data types, to serve as a basis for intelligent computational support of knowledge discovery workflow composition. A formal capture of knowledge discovery tasks can then be used to improve repeatability of experiments and to enable reasoning on the results to facilitate reuse of workflows and results.

An intelligent Service-oriented Knowledge Discovery (SoKD) approach, proposed in this paper, aims at developing a third generation knowledge discovery framework [11] and its implementation that is intended to address the challenges discussed above. A practical implementation of the proposed third-generation knowledge discovery platform is based on extending an existing data mining platform Orange [6]. We named it Orange4WS. The third-generation data mining paradigm shift implies the need for a substantially different knowledge discovery platform, aimed at supporting human experts in scientific discovery tasks. In comparison with the current publicly available data mining platforms (e. g. Weka [32], KNIME [5], Orange [6]), our platform should provide an intelligent tool which makes use of a knowledge discovery ontology, supported by planning and ontological reasoning, based on top of a rich collection of data processing and mining components as well as data and information sources provided by local processing components and web services.

The paper is structured as follows. Section 2 presents some of the related work, Section 3 present our work in upgrading an existing data mining tool in order to be suitable for the SoKD methodology. Sections 4 and 5 upgrade the implemented solution by introducing ontologies, annotations and planning. Section 6 demonstrates some of the abilites of the platform on a text mining use case. Section 7 concludes with a summary and plans for further work.

2 Related Work

In this section we examine previous work related to the key ingredients of our framework: knowledge discovery domain formalization for workflow construction and reuse, workflow editing and execution environment and service oriented architecture for knowledge discovery.

There exist several systems using a formal representation of data mining (DM) operators for automatic workflow composition and ranking including IDEA [4], NExT [3] and KDDVM [8]. However all these systems focus only on propositional data mining and produce only linear or tree workflows, whereas we focus on complex relational data mining and text mining and our workflows are directed acyclic graphs.

Other efforts to provide a systematic formalization of the DM tasks include projects MiningMart [21] and DataMiningGrid [27] and system described in [18]. The first two focus on mining propositional patterns from data stored in a relational database. None of the systems provide means for automatic workflow creation. In parallel to our work, the OntoDM [22] ontology is being developed. A principled top-down approach was adopted to the development of OntoDM

aiming at its maximal generality. Given the complexity of the domain subject to be modeled, the ontology is currently not sufficiently refined for purposes of workflow construction. Also, unlike our ontology, OntoDM is not compatible with OWL-S.

Also relevant are solutions to the problem of web service composition in the framework of planning. The work of [17] relies on computing a *causal link matrix* for all the available services. However we work with a more general, non-linear notion of a plan. Work reported in [26], [20] and [19] translate an OWL description to a planning formalism based on PDDL. While the work presented in [20] and [19] use classical STRIPS planning, in [26], Hierarchical Task Network (HTN) planning is employed. HTN is not applicable in our framework not constrained to tree-based task decomposition. The approach presented in [19] and [20] uses a reasoner in the pre-processing phase; we make a step beyond by integrating a reasoning engine directly with the planner. Planning directly in description logics is addressed in [13]. Currently the algorithm can only deal with DL-Lite descriptions with reasonable efficiency.

Construction of analytic workflows has attracted a lot of development in recent years. The most well-known systems include Triana [29], a workflow environment for P2P and Grid containing a system for integrating various types of middleware toolkits, and Taverna [7] environment for workflow development and execution. Although systems such as Magallanes [23] offer limited support for automatic workflow construction by using a data type hierarchy and breadth-first search to produce a given target data type as output, they are not able to use a knowledge discovery ontology, and are not general enough to support arbitrary data mining tasks. Also, their abilities for automatic workflow construction are limited to the bioinformatics domain.

There has been some work on workflows for distributed data mining using a service-oriented architecture, e.g., [12] and [1]. However most of the systems focus on demonstrating the feasibility of service-oriented approach for distributed data mining with regard to parallelization and distributed data sources. Nevertheless, none of these approaches enables automated DM workflow construction.

Also relevant for our work is Weka4WS [28], which is a framework that extends the Weka toolkit to support distributed data mining on Grid. Weka4WS user interface supports the execution of both local and remote data mining tasks. However, only native Weka components and extensions are available and the framework does not support arbitrary web services, which can be found on the internet.

3 Orange4WS Platform

This section briefly describes the structure and design of the presented SoKD software platform. We explain and comment our decisions about technologies and software tools that were used. The main part of the section is a description of the design of Orange4WS and the accompanying toolkit for producing new web services.

3.1 Design and Implementation

The proposed software solution, named Orange4WS, is built on top of two open-source scientific community driven projects: the Orange data mining framework

[6], which provides a range of preprocessing, modeling and data exploration techniques, and the Python Web Services project[1], which provides libraries for the development of web services using the Python programming language[2] by implementing various protocols including SOAP, WSDL, etc.

The Orange framework features visual programming, achieved by graphical composition of processing components into workflows, which are essentially visual representations of complex procedures. Orange provides Orange Canvas where such componens (called Orange Widgets) can be connected, adjusted, and executed. Orange4WS extends this model by introducing web services and knowledge discovery ontology (this is discussed in Section 4).

In contrast with other freely available workflow environments such as Weka, Taverna, Triana, KNIME, RapidMiner, etc. the Orange (Orange4WS) framework offers a rather unique combination of features. These features are:

- a large collection of data mining and machine learning algorithms, efficiently implemented in C++
- three-layer architecture: C++, Python, Orange and Orange4WS widgets
- a collection of very powerful yet easy to use data visualization widgets
- simplicity of Orange's workflows.

Built on top of Orange, Orange4WS consists of two layers: the supporting layer, which deals with technical details of web services (information extraction, execution, error reporting, data transfer and transformation, code generator) and the upper layer, which uses low-level functionalities to enable web services as building blocks (widgets) of the Orange Canvas. The structure of Orange4WS is summarized in Figure 1a. The invocation of web services, data transformation, message passing, error handling, etc. are hidden and can be summarized from the user's perspective as a normal widget operation: (1) receiving data, (2) widget internal processing, and (3) outputting processed data. Thus, the user is completely isolated from all web-service related technical details which greatly simplifies the creation, understanding, and execution of workflows.

If a workflow is constructed manually, the Orange Signal manager, which lives inside the Orange Canvas, enables or disables the connectivity of Orange and Orange4WS widgets with respect to their input/output types (automated workflow construction is discussed in Section 5). In order to construct an Orange4WS workflow, the user only needs to instantiate all required widgets by clicking on their icons, position them in the right order according to his/her objective, and connect them with the help of signal manager, which prevents invalid connections according to the type information. In comparison with the popular Taverna workbench [15], Orange4WS workflows are simpler to construct and much easier to understand because of the higher level of abstraction when using web services. Also, Orange4WS workflows are better suited for data mining and knowledge discovery

[1] http://pywebsvcs.sourceforge.net/

[2] Use of the pythonic branch of the WSO2 web service framework (based on Apache Axis2/C) and Suds SOAP client for Python is planned in the future.

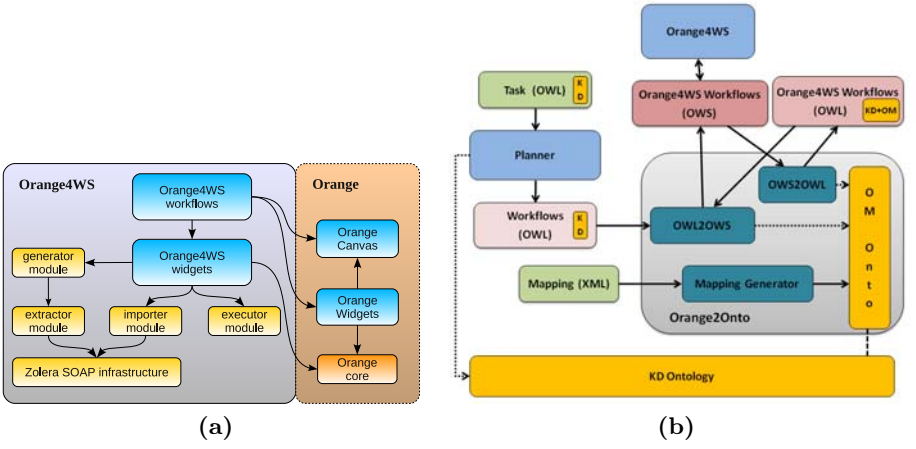


Fig. 1. The strucure of the Orange4WS platform built on top of the Orange data mining toolkit (a) and a detailed overview of the integrated framework for annotations and planning (b) which is described in more details in Section 5

in general since they can be used in combination with a wide range of machine learning algorithms, data mining algorithms, and visualization methods, already available in Orange.

3.2 Production of New Web Services

A separate part of our service-oriented knowledge discovery platform is a package of tools which ease the production of new services. These tools closely follow the general *"WSDL first"* design principle [10] which promotes clearly designed, interoperable and reusable services by separating the design of interfaces from the actual logic. Essentially, our tools extend the Python language framework by using the Python Web Services package, enchanced with multiprocessing capabilities, security, logging and other related functionalities. Most notably, our web service production tools are able to circumvent the well known global interpreter lock (GIL) of the Python interpreter by employing the *multiprocessing* module which enables full utilization of today's multiprocessor computers. Using these tools, any code can easily be transformed into a SOAP web service and used as an ingredient for Orange4WS workflow composition (or in any other workflow environment capable of using web services).

4 Knowledge Discovery Ontology

To provide the proposed knowledge discovery platform with semantics, we are using the *Knowledge Discovery Ontology* (KD ontology, for short) [31]. The ontology defines relationships among the ingredients of knowledge discovery

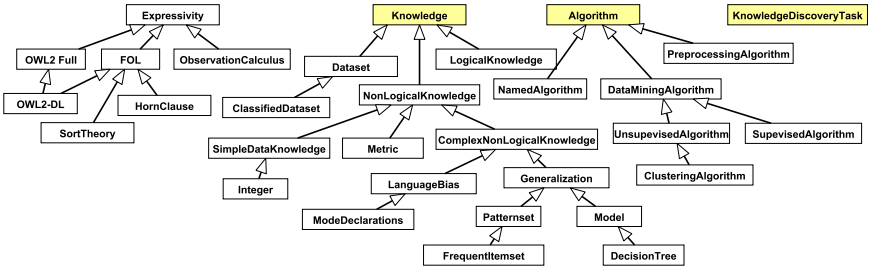


Fig. 2. Part of the top level structure of the KD ontology with subclass relations shown through arrows

scenarios, both declarative (various knowledge representations) and algorithmic. We are using the ontology to enable the workflow planner to reason about which algorithms can be used to produce the results required by a specified knowledge discovery task and to query the results of the knowledge discovery tasks. In addition, the ontology can also be used for automatic annotation of manually created workflows facilitating their reuse.

An illustrative part of the top level structure of the ontology is shown in Figure 2. The three core concepts are: *knowledge*, capturing the declarative elements in knowledge discovery, *algorithms*, which serve to transform knowledge into (another form of) knowledge, and *knowledge discovery task*, which describes a task the user wants to perform mainly by specifying available data and knowledge sources and desired outputs. The ontology is implemented in the semantic web language OWL-DL[3]. The core part of the KD ontology currently contains around 150 concepts and is available online[4]. The structure of workflows is described using OWL-S[5]. For the purposes of this work we have extended the ontology with concepts describing the ingredients of text mining tasks.

In the following subsections we describe the concepts of *knowledge* and *algorithm* in more detail and provide information on the annotation of algorithms available locally in the Orange4WS toolkit and in the LATINO library.

4.1 Knowledge

Any declarative ingredient of the knowledge discovery process such as datasets, constraints, background knowledge, rules, etc. are instances of the `Knowledge` class. In data mining, many knowledge types can be regarded as sets of more elementary pieces of knowledge [9], e.g., first-order logic theories consist of formulas. This structure is accounted for through the property `contains`, so e.g. a first-order theory `contains` a set of first-order formulas.

[3] http://www.w3.org/TR/owl-semantics/
[4] http://krizik.felk.cvut.cz/ontologies/2008/kd.owl
[5] http://www.w3.org/Submission/OWL-S/

4.2 Algorithms

The notion of an algorithm involves all executable routines used in a knowledge discovery process, ranging from inductive algorithms to knowledge format transformations. Any algorithm turns a knowledge instance into another knowledge instance, e.g. inductive algorithms will typically produce a `Generalization` instance out of a `Dataset` instance. The concept of algorithm is central to the work presented in this paper. The `Algorithm` class is a base class for all algorithms, like Apriori (algorithm for association rule induction implemented in Orange) and GenerateBows (algorithm implemented in LATINO for constructing bag of words representation from a collection of documents, described in the example below). For this work we have refined the hierarchy of fully defined classes, like `DecisionTreeAlgorithm` or `DataPreprocessingAlgorithm` for fine-grained categorization of data mining algorithms according to their functionality. The hierarchy of algorithms allows for the formulation of additional constraints on the workflows, e.g., there should be at most two preprocessing algorithms on each branch of the workflow. Each algorithm configuration is defined by its input and output knowledge specifications and by its parameters.

4.3 Annotating Algorithms

The KD ontology was used to annotate most of the algorithms available in the Orange toolkit. More than 60 algorithms have been annotated so far. We have also annotated the components of LATINO using the mapping described in [16]. As an example we present a definition of the `GenerateBows` algorithm which can be applied to a collection of documents and produces a bag of words representation of these documents. The settings are quite complex, therefore they are provided as a single input object. The definition in the description logic notation using the extended ABox syntax [2] is shown in Figure 3.

The annotated algorithms also served as case studies to validate and extend the KD ontology, therefore developing a procedure for semi-automatic annotation is a subject for future work.

$$
\begin{aligned}
\{\texttt{GenerateBows}\} &\sqsubseteq \texttt{NamedAlgorithm} \\
&\sqcap \exists\, \texttt{output} \cdot \{\texttt{GenerateBows-O-Bows}\} \\
&\sqcap \exists\, \texttt{input} \cdot \{\texttt{GenerateBows-I-Docs}\} \\
&\sqcap \exists\, \texttt{input} \cdot \{\texttt{GenerateBows-I-Settings}\} \\
\{\texttt{GenerateBows-I-Docs-Range}\} &\equiv \texttt{isRangeOf} \cdot \{\texttt{GenerateBows-I-Docs}\} \\
&\equiv \texttt{DocumentCollection} \\
\{\texttt{GenerateBows-O-Bows-Range}\} &\equiv \texttt{isRangeOf} \cdot \{\texttt{GenerateBows-O-Bows}\} \\
&\equiv \texttt{BowSpace}
\end{aligned}
$$

Fig. 3. A definition of the `GenerateBows` method in the description logic notation using the extended ABox syntax

5 Automated Workflow Construction

The focus of this section is on automatic construction of abstract workflows of data mining algorithms. We treat the automatic workflow construction as a planning task, in which algorithms represent operators and their input and output knowledge types represent preconditions and effects. However, since the information about the algorithms, knowledge types and the specification of the knowledge discovery task is encoded through an ontology, we implemented a planning algorithm capable of directly querying the KD ontology using a reasoner. The Pellet[6] reasoner was used. The main motivation for using Pellet was its ability to deal with literals, availability in Protégé[7], which we used for ontology development, and processing of SPARQL-DL [25] queries.

In the next sections we present an enhanced version of the algorithm described in [31], which exploits the algorithm hierarchy for planning at multiple abstraction levels. Furthermore, a post-processing step using the ontology was added for a more user-friendly presentation of KD workflows, in case the task is weakly constrained and thus a relatively large number of workflows is generated.

5.1 Exploiting Algorithm Hierarchy

The planning algorithm used to generate abstract workflows automatically is based on the Fast-Forward (FF) planning system [14]. We have implemented the basic architecture of the FF planning system consisting of the enforced hill climbing algorithm and the relaxed GRAPHPLAN. Since the planning problem in workflow construction contains no goal ordering, no mechanisms for exploiting goal ordering were implemented.

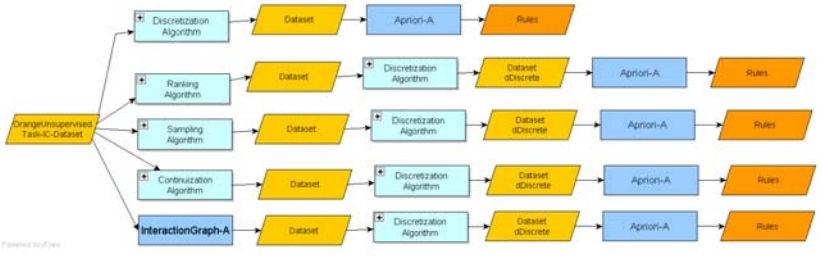


Fig. 4. An example of workflows for discovering association rules in Orange

The planner obtains neighboring states during enforced hill-climbing by matching preconditions of available algorithms with currently satisfied conditions. Each matching is conducted during the planning time via posing an appropriate SPARQL-DL query to the KD ontology. We have enhanced the algorithm in two

[6] http://clarkparsia.com/pellet/
[7] http://protege.stanford.edu/

ways: a hierarchy of algorithms based on defined classes and input/output specifications computed and in searching for neighboring states the planner exploits the algorithm hierarchy.

A hierarchy of algorithms is inferred before the actual planning. It needs to be recomputed only when a new algorithm is added to the ontology. The hierarchy of algorithms is based on the inputs and outputs of the algorithms and on the defined algorithm classes such as `PreprocessingAlgorithm`. It holds that $A_j \sqsubseteq A_i$ if for every input of I_{ik} A_i there is an input I_{jl} of algorithm A_j such that range of $I_{ik} \sqsubseteq I_{jl}$. An algorithm $A_i \equiv A_j$, if $A_j \sqsubseteq A_i$ and $A_i \sqsubseteq A_j$. The subsumption relation on algorithms is used to construct a forest of algorithms with roots given by the explicitly defined top-level algorithm classes e.g. `DataPreprocessingAlgorithm`.

```
task - instance of KnowledgeDiscoveryTask
maxSteps - max length of the workflow
constr - additional constraints
generateWorkflows(task, maxSteps, constr):
   classify KD ontology;
   algs := {instances of NamedAlgorithm};
   algforest := inferAlgorithmHierarchy(algs);
   workflows := runPlanner(task, algforest, maxSteps);
   atomicW := expandWorkflows(workflows, algforest);
   filteredW := filterWorkflows(atomicW, constr);
```

Fig. 5. A skeleton of the procedure for workflow composition using the KD ontology

The planning algorithm was adapted so that in the search for the next possible algorithm it traverses the forest structure instead of only a list of algorithms and considers a set of equivalent algorithms as a single algorithm. Currently, only constraints on repetition of some kind of algorithms (defined by a class or set of classes in the KD ontology) in a linear part of the workflow are built into the planner. The additional constraints on workflows are used only for filtering of the generated workflows during post-processing (procedure `filterWorkflows`). Workflows for all the members of an equivalence set are generated using the procedure `expandWorfklows`. The information about algorithms subsumption is also used in presenting of workflows to the user. An overview of the whole procedure for workflow generation is shown in Figure 5.

The generated workflows are presented to the user using an interactive visualization, which enables the user to browse the workflows from the most abstract level to specific combination of algorithm instances. The workflows with the smallest number of steps are presented first. An example of a set of workflows generated for discovering association rules in Orange4WS is in Figure 4.

5.2 A Framework for Workflow Execution in Orange4WS

An overview of the framework is shown in Figure 1b. The module Orange2Onto, which acts as an interface between Orange4WS and ontology representation does

not work directly with internal representation of Orange4WS, but it works with the OWS format used in the standard Orange distribution to store workflows in the XML format.

In order to capture formally the mapping between the internal Orange4WS representation and the representation of algorithms using the KD ontology, the Orange-Map (OM) ontology was developed defining templates for mapping of algorithms, data and parameters. The OM ontology is then used for converting the automatically generated workflows into the Orange representation. In order to facilitate the creation of the mapping for new algorithms, the mapping can be specified using an XML file. The corresponding instances in the ontology are then generated automatically.

Annotations of Orange4WS workflows containing algorithms not annotated using the KD ontology can also be created automatically. The missing information about input/output types of the algorithms is then either deduced from the links with annotated algorithms or considered to be some `Knowledge` expressed as string. The annotations of such workflows can therefore be used for some querying and repeating of experiments, however the generated annotation of the unknown algorithm is not suitable for planning.

The procedures for converting Orange4WS representation to OWL and vice versa are implemented in Python using JPype[8] to call the Jena[9] ontology API implemented in Java.

6 A Text Mining Use Case

This use case is built upon text mining web services, available from the LATINO[10] text mining library, which provides a range of data mining and machine learning algorithms with the emphasis on text mining, link analysis, and data visualization. Recently, its functionalities have become available as web services to ease the use of the library and consequently broaden its user community.

The goal of this use case is to produce a graph of terms, which could potentially give insight into relations between biological, medical and chemical terms, relevant to the subject of the query. Additionaly, the Pathfinder algorithm was applied to reduce the complexity of the term graph. The presented use case requires five LATINO services in combination with the PubMed search service, local Orange graph visualization tool, and the Pathfinder service. Therefore, the use case demonstrates the need for a service-oriented platform able to combine publicly available data repositories (PubMed), specialized algorithms (Pathfinder), third-party data analysis tools (LATINO) and powerful local visualization components (Orange graph visualizer).

The LATINO algorithms, used to build the workflow, were annotated manually using their WSDL description to enable the planner to automatically build

[8] http://jpype.sourceforge.net/
[9] http://jena.sourceforge.net/
[10] http://sourceforge.net/projects/latino

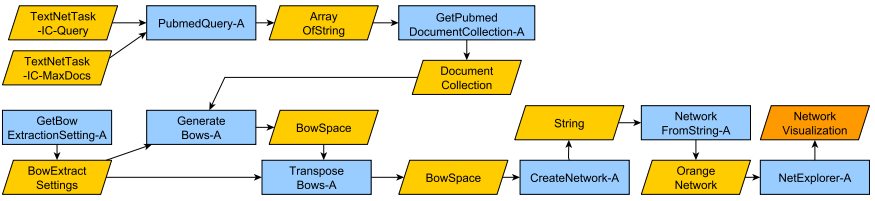


Fig. 6. An example of an automatically generated workflow for the task of visualizing the query graph

workflows using these annotated components. However, the number of alternative workflows which can be produced, is currently quite small due to use of complex knowledge types and specialized algorithms (for example, there is currently only one procedure for obtaing sparse vectors from text). An automatically discovered workflow is presented in Figure 6.

The Pathfinder algoritm [24] is an example of a network prunning algorithm which was developed as a tool in cognitive science to determine the most important links in a network. The prunning idea of the algorithm is based on the triangle inequality which is extended to all paths in a network: the direct distance between two nodes must be less than or equal to the sum of all weights of every path between these two nodes. The Pathfider algorithm thus elliminates all links which violate the extended triangle inequality. More specifically, to calculate the distance between two nodes along in a path in the graph the Pathfinder algorithm uses the generalized distance (Minkowski distance) of order p which is defined as $(\sum_{i=1}^{n} |x_i - y_i|^p)^{\frac{1}{p}}$. For example, $p = 1$ gives Manhattan distance, $p = 2$ gives Euclidean distance while $p = \infty$ gives $max(x, y)$. The

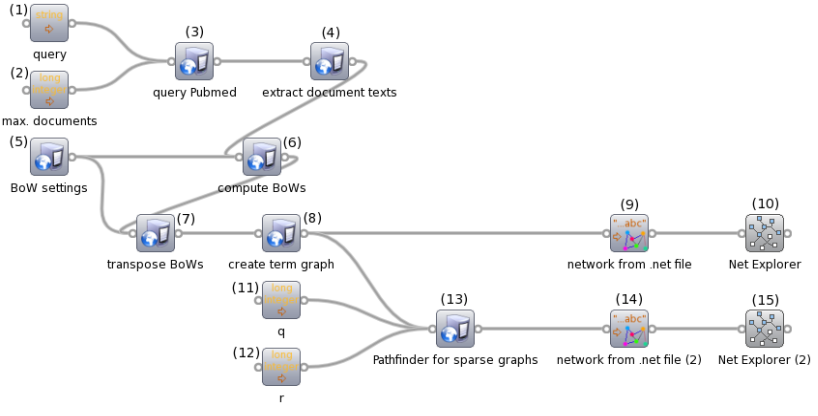
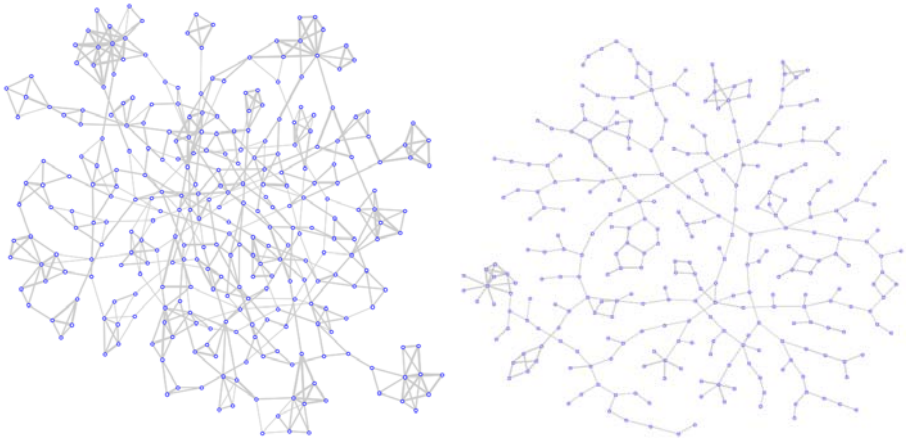


Fig. 7. The implementation of a workflow for visualization and prunning of a term graph in Orange4WS. Components numbered 3, 4, 5, 6, 7, 8 and 13 are web services, components 1, 2, 9, 11, 12 and 14 are Orange4WS supporting widgets, components 10 and 15 are native Orange graph visualizers.



(a) **(b)**

Fig. 8. The results of the execution of the workflow from Figure 7. Figure 8a shows the term graph obtained by querying PubMed using the term "migraine+magnesium". Figure 8b shows the pathfinder network obtained by applying the Pathfinder algorithm on the term graph. Because of the space limitations names of vertices and weights of edges are not shown. Widths of edges represent their weights, respectively.

second parameter of the algorithm (q) controls the maximal length of all examined paths.

An implementation of the Pathfinder algorithm and its much more efficient variant for sparse graphs [30] was transformed into a SOAP web service to enable its use in a service-oriented workflow environment. A concrete realization of the workflow in the Orange4WS environment is shown in Figure 7.

PubMed is queried with a query string and maximal number of documents (components 1, 2, and 3). It returns a collection of IDs of relevant documents. Then, obtained IDs are used to collect titles, abstracts and keyword of these documents (component 4). Afterwards, bag-of-words (BoW) sparse vectors are created from the collection of words (component 6). To simplify setting the parameters for unexperienced users there is a service providing a suitable set of default values which can be used as an input to the web service which constructs BoW vectors (component 5). Then, BoW vectors are transposed and a term network is created (we omit the details how this graph is produced as it is out of the scope of this paper. More details are available in [30]). Component 8 produces a network in the widely used Pajek's .net format[11], which is finally loaded into Orange's native graph structure and visualized with the `Net explorer` widget (component 10). The lower branch of the workflow sends the term graph into the Pathfider algorithm web service which simplifies the graph by removing edges violating the extended triangle inequality. For example, the query string

[11] http://pajek.imfm.si/doku.php

"migraine+magnesium" with the limit of 30 documents produced a graph shown in Figure 8a. The simplified graph, obtained by applying the Pathfinder algorithm is shown in Figure 8b.

7 Conclusions

This paper proposes a third-generation knowledge discovery framework and its implementation named Orange4WS. Based on a data mining toolkit (Orange) that supports execution of workflows of processing components, our new platform upgrades its capabilities by transparent integration of web services. As web services are an extremely versatile and powerful concept which is becoming more and more popular, we believe their use in knowledge discovery will increase rapidly.

We have added semantic capabilities to the framework by proposing a methodology for integrating semantic annotation and planning into a data mining platform by means of an ontology describing a mapping between KD representation and native representation of the data mining platform. This methodology was primarily targeted on Orange, however it could be modified to other data mining platform. We have developed a planner, which exploits the hierarchy of algorithms annotated using the KD ontology. The construction of algorithm hierarchy is time consuming, but it needs to be recomputed only when a new algorithm is added to the ontology. Moreover, the hierarchy can also be exploited in the presentation of the workflows to the user. The presented implementation and methodology were demonstrated on a text mining use case which demostrates some of the abilities of our platform and versatility of web services by means of integration of different knowledge discovery systems and algorithm implementations.

In future work we will explore adding means for semantic web service discovery and semi-automatic annotation. The planner will also be a subject of future improvements as we plan to incorporate the ability of satisfying user-defined constraints and preferences.

References

1. Ali, A., Rana, O., Taylor, I.: Web services composition for distributed data mining. In: Proc. of the 2005 IEEE Int. Conf. on Parallel Processing Workshops. IEEE, Los Alamitos (2005)
2. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P. (eds.): The Description Logic Handbook, Theory, Implementation and Applications. Cambridge University Press, Cambridge (2003)
3. Bernstein, A., Deanzer, M.: The NExT system: Towards true dynamic adaptions of semantic web service compositions (system description). In: Franconi, E., Kifer, M., May, W. (eds.) ESWC 2007. LNCS, vol. 4519, pp. 739–748. Springer, Heidelberg (2007)
4. Bernstein, A., Provost, F., Hill, S.: Toward intelligent assistance for a data mining process: An ontology-based approach for cost-sensitive classification. IEEE Trans. on Knowledge and Data Engineering 17(5), 503–518 (2005)

5. Berthold, M.R., Cebron, N., Dill, F., Gabriel, T.R., Kötter, T., Meinl, T., Ohl, P., Sieb, C., Thiel, K., Wiswedel, B.: KNIME: The Konstanz information miner. In: Studies in Classification, Data Analysis, and Knowledge Organization (GfKL 2007). Springer, Heidelberg (2007)
6. Demšar, J., Zupan, B., Leban, G.: Orange: From experimental machine learning to interactive data mining. White Paper (2004)
7. DeRoure, D., Goble, C., Stevens, R.: The design and realisation of the myExperiment virtual research environment for social sharing of workflows. Future Generation Computer Systems 25, 561–567 (2008)
8. Diamantini, C., Potena, D., Storti, E.: KDDONTO: An ontology for discovery and composition of KDD algorithms. In: SoKD: ECML/PKDD 2009 Workshop on Third Generation Data Mining: Towards Service-oriented Knowledge Discovery, pp. 13–24 (2009)
9. Džeroski, S.: Towards a general framework for data mining. In: Džeroski, S., Struyf, J. (eds.) KDID 2006. LNCS, vol. 4747, pp. 259–300. Springer, Heidelberg (2007)
10. Erl, T.: Service-Oriented Architecture: Concepts, Technology, and Design. Prentice-Hall, Englewood Cliffs (2006)
11. Finin, T., Gama, J., Grossman, R., Lambert, D., Liu, H., Liu, K., Nasraoui, O., Singh, L., Srivastava, J., Wang, W.: National science foundation symposium on next generation of data mining and cyber-enabled discovery for innovation (NGDM 2007): Final report (2007)
12. Guedes, D., Meira, W.J., Ferreira, R.: Anteater: A service-oriented architecture for high-performance data mining. IEEE Internet Computing 10(4), 36–43 (2006)
13. Hoffmann, J.: Towards efficient belief update for planning-based web service composition. In: Proc. of ECAI 2008, pp. 558–562 (2008)
14. Hoffmann, J., Nebel, B.: The FF planning system: Fast plan generation through heuristic search. Journal of Artificial Intelligence Research 14, 253–302 (2001)
15. Hull, D., Wolstencroft, K., Stevens, R., Goble, C., Pocock, M., Li, P., Oinn, T.: Taverna: a tool for building and running workflows of services. Nucleic Acids Research 34, 729–732 (2006)
16. Kalyanpur, A., Jiménez Pastor, D., Battle, S., Padget, J.A.: Automatic mapping of OWL ontologies into Java. In: Proc. of SEKE 2004, pp. 98–103 (2004)
17. Lécué, F., Delteil, A., Léger, A.: Applying abduction in semantic web service composition. In: Proc. of the ICWS 2007, pp. 94–101 (2007)
18. Li, Y., Lu, Z.: Ontology-based universal knowledge grid: Enabling knowledge discovery and integration on the grid. In: Proc. of the 2004 IEEE Int. Conf. on Services Computing (2004)
19. Liu, Z., Ranganathan, A., Riabov, A.: A planning approach for message-oriented semantic web service composition. In: Proc. of the Nat. Conf. on AI, vol. 5(2), pp. 1389–1394 (2007)
20. Klusch, M., Gerber, A., Schmidt, M.: Semantic web service composition planning with OWLS-XPlan. In: Procs of 1st Intl. AAAI Fall Symposium on Agents and the Semantic Web (2005)
21. Morik, K., Scholz, M.: Web services composition for distributed data mining. In: Proc. of the International Conference on Machine Learning, pp. 47–65 (2004)
22. Panov, P., Džeroski, S., Soldatova, L.N.: OntoDM: An ontology of data mining. In: Proceedings of the IEEE ICDM Workshops 2008, pp. 752–760 (2008)
23. Rios, J., Karlsson, J., Trelles, O.: Magallanes: a web services discovery and automatic workflow composition tool. BMC Bioinformatics 10(1) (2009)
24. Schvaneveldt, R.W., Dearholt, D.W., Durso, F.T.: Graph theoretic foundations of pathfinder networks. Computers and Mathematics with Applications (1988)

25. Sirin, E., Parsia, B.: SPARQL-DL: SPARQL query for OWL-DL. In: Proc. of the OWLED 2007 Workshop on OWL: Experiences and Directions (2007)
26. Sirin, E., Parsia, B., Wu, D., Hendler, J., Nau, D.: HTN planning for web service composition using SHOP2. Journal of Web Semantics 1(4), 377–396 (2004)
27. Stankovski, V., Swain, M., Kravtsov, V., Niessen, T., Wegener, D., Kindermann, J., Dubitzky, W.: Grid-enabling data mining applications with DataMiningGrid: An architectural perspective. Future Generation Computer Systems 24(4), 259–279 (2008)
28. Talia, D., Trunfio, P., Verta, O.: Weka4WS: A WSRF-enabled Weka toolkit for distributed data mining on grids. In: Jorge, A.M., Torgo, L., Brazdil, P.B., Camacho, R., Gama, J. (eds.) PKDD 2005. LNCS (LNAI), vol. 3721, pp. 309–320. Springer, Heidelberg (2005)
29. Taylor, I., Shields, M., Wang, I., Harrison, A.: The Triana workflow environment: Architecture and applications. In: Taylor, I., Deelman, E., Gannon, D., Shields, M. (eds.) Workflows for e-Science, pp. 320–339. Springer, Heidelberg (2007)
30. Vavpetič, A., Batagelj, V., Podpečan, V.: An implementation of the pathfinder algorithm for sparse networks and its application on text networks. In: Proceedings of the 12th international multiconference Information Society (IS 2009), pp. 236–239 (2009)
31. Žáková, M., Křemen, P., Železný, Lavrač, N.: Automatic knowledge discovery workflow composition through ontology-based planning. IEEE Trans. Automation Science and Engineering (2010)
32. Witten, I.H., Frank, E.: Data Mining: Practical machine learning tools and technique, 2nd edn. Morgan Kaufmann, San Francisco (2005)

Workflow Composition and Enactment Using jORCA

Johan Karlsson[1,2], Victoria Martín-Requena[1], Javier Ríos[1], and Oswaldo Trelles[1]

[1] Computer Architecture Department; University of Malaga; Campus de Teatinos, 29071 Málaga, Spain
[2] Fundación IMABIS, Málaga, Spain
{tjkarlsson,vickymr,jriosp,ortrelles}@uma.es

Abstract. Support for automated service discovering and workflow composition is increasingly important as the number of Web Services and data types in bioinformatics and biomedicine grows. jORCA is a desktop client able to discover and invoke Web Services published in different types of service metadata repositories. In this paper, we report that jORCA is now able to discover, compose, edit, store, export and enact workflows. As proof of concept, we present a case study which re-creates a published workflow to demonstrate new functionality in jORCA, starting from service discovery, workflow generation and refinement; to enactment, monitoring and visualization of results. The system has been exhaustively tested and documented and is freely available at http://www.bitlab-es.com/jorca.

1 Introduction

Bioinformatics is strongly based on the Web since it can be used to deploy bioinformatics tools and provide universal access to distributed resources. This is important since many applications in bioinformatics require access to constantly updated databases and have large computational requirements, something not possible to maintain in small laboratories. However, a weakness of using Web tools –in the general sense of applications available in the Web- is that they can differ in their descriptions, invocation protocols and data formats, which represents a barrier to service interoperability. Moreover, the proliferation of bioinformatics services makes it difficult to locate the appropriate service or set of services for data processing. Thus, discovery, composition (workflow generation) and invocation of services become complicated tasks. In this scenario, Web Services appears as a potential solution since they are "designed to support interoperable machine-to-machine interaction over a network" [20].

Several clients have been developed to assist in the utilization of WS. Most of these clients are oriented to specific data models, service metadata and protocols. Gbrowse [22], MOWServ [31], Dashboard [1], Seahawk [7] are well-known clients but they are limited to work with BioMOBY [21] Web Services. BioCatalogue [6], the European registry for biological Web Services, stores metadata for SOAP (for example BioMOBY and SoapLab) and REST-style services. In the workflow composition and enactment area, Remora [3] and BioWEP [18] are also specific for BioMOBY services. The current reference software in bioinformatics for workflow composition and enactment is Taverna [14] but this application has a steep learning

T. Margaria and B. Steffen (Eds.): ISoLA 2010, Part I, LNCS 6415, pp. 328–339, 2010.

rate [8]. A full comparison by requisites of jORCA and these clients can be found in the Supplementary material in [12] and jORCA's web page documents.

To facilitate seamless integration of WS, we developed jORCA [12] which is a desktop client based on a uniform representation of different web resources enabling service invocation and advanced control over invocations. In the same line, to facilitate bioinformatics tools exploitation, we developed Magallanes [17]; a versatile, platform-independent Java library of algorithms to discover appropriate WS associated to specific data types, enabling the development of clients for service discovery. A second, important, feature of Magallanes is its ability to connect compatible WS into pipelines (simple workflows). Pipelines are chains of processing elements (WS), arranged so that the output of each element is the input of the next. Executing pipelines consist in executing services sequentially. Support for manual and automatic execution of pipelines should be provided to help carrying out complex tasks that involves executing a set of tools. Magallanes is completely integrated with jORCA.

In this document, we describe the extension of jORCA with additional functionality, including:

- Workflow enactment, including monitoring of enactments in progress
- Extended support for workflow composition by means of drag and drop from the history of web-service invocations.

A case study is also presented and elaborated in Supplementary Material to illustrate how to reproduce a previously published workflow [11] for clustering of co-regulated genes to obtain a hierarchical multiple sequence alignment that displays the most relevant common features of those genes at sequence level. The case study will demonstrate workflow functionality in jORCA, starting with the discovery of services and datatypes which are used as input to service composition (workflow generation). The generated service composition contains several alternative paths of which the user can select the more suitable. The completed workflow is enacted within jORCA which is also used to visualize results.

This paper is organized as follows: in this first section, we introduce the area of web-services in bioinformatics and jORCA. In Section 2, we describe the overall architecture of jORCA and describe the field of workflow composition and the approach of jORCA. In Section 3, we describe how jORCA can compose the workflow from [11]. This workflow was originally manually composed and we show how the new functionality simplifies greatly the task of workflow composition. Section 4 contains a brief comparison with other approaches for composition of BioMOBY services and a discussion of other issues. Finally, Section 5 concludes the paper.

2 System and Methods

jORCA architecture is described in [12]. jORCA combines different software components specialized for specific tasks (see Fig. 1). The lower layer of jORCA uses the framework from [16] to represent services metadata uniformly from the point of view of jORCA. Software components (accesses) are used to map the different service models into a single virtual representation and specific workers handle the specific

service invocations (specified by their protocols). New functionality can be added to jORCA with plugins, which can request an area in the central pane of the graphical user interface with full access to service and datatype information. Finally, the top layer deals with the graphical user interface.

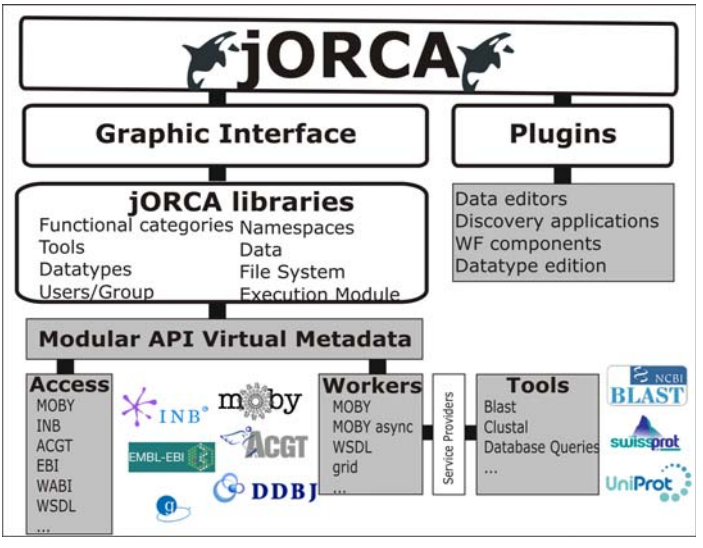


Fig. 1. jORCA architecture. The Modular API [16] offers a uniform view of different repositories. Access modules are used to map different repositories and specific workers manage the different invocation protocols. A plug-in interface allows jORCA to extend its functionality.

Discovery Methods

The number of registered Web Services in repositories or described by individual WSDL files make service discovery a challenging tasks (see Table 1 for examples). In general, a discovery process aims to segregate a set of services or data types that satisfy a given number of requirements from a larger pool of available resources (services). We address this problem in Magallanes, which has been fully integrated with jORCA. Apart from the traditional discovering methods, mostly based on syntactic text-search engines and filtering mechanisms (also covered by Magallanes using keywords which are matched against service descriptions retrieved from the different repositories), Magallanes can perform inexact searches based on a scoring system and provides 'did you mean…?' suggestions to manage approximated matches.

Extending jORCA with Workflow Functionality

Automatic *workflow generation*, also called automatic *service composition*, aims to automate (to various degrees) the task of connecting services. In general, the problem addressed in this area is; *given two sets of data types (representing inputs and outputs respectively), find the optimal set of inter-connected services where the input data types of the initial service(s) are of the specified input data types and the output data*

types of the last services are of the specified output data types. Two services can be inter-connected if the output of one is compatible with the input of the other.

Usually the optimum set is determined by degree of semantic closeness and non-functional metadata (QoS). For example, workflow generation algorithms typically favor services whose input/output data types are semantically close (similar) to the requested data type. Examples of non-functional metadata include level of availability, error rates, cost (expected CPU time or data transfer time).

If the number of services and or data types is low, the process of building workflows can even be manual, provided that the services have sufficient textual descriptions. However, when the number of services reach a certain level, support in connecting services becomes essential.

Such support can be either semi-automatic, interactively giving advice regarding suitable services for each step in the workflow construction or fully automatic where the human workflow builder only provides input and output data descriptions and the algorithm directly generates complete workflows.

Generic approaches to service composition consider all possible solutions and local approaches consider only sub-sets of the possible solutions. A novel approach to service composition using the BioMOBY-S framework is described in [4]. Several improvements intended to limit the search space are described and included in the algorithm. In BioMOBY-S, several ontologies are used in service descriptions. In particular, data type ontology is shared between all services. In the ontology, data types are related to other data types; in particular, a data type can extend definitions of other data types or be composed of other data types (i.e. complex data types). These relations can be recursive, meaning that composite data types can extend other data types (super classes) and contain other complex data types (components). The service composition algorithm in [4] considers not only direct compatibility and polymorphism (compatibility by inheritance), but also includes services where the input/output has *"contains"*-relations directly or indirectly (recursively) with the required data type. For example, this means that compatibility is evaluated not only for the data type itself but also for all other data types that the data type is composed of. Results are further ranked by their popularity as measured by requests to the central registry and distance between data types in the ontology.

In simplest terms, the automatic WF-builder in jORCA (through Magallanes) proceeds to identify all the services that produce a target data type as output. All the data types used as input for such services are used as a target in the next step. If there are several options, jORCA will display these options and let the user select the appropriate option. When there are many options, the graph can be difficult to explore. In this case, the default view is the compact mode where branches of alternative services are shown as stacks. The default service selected from each stack is based on the feedback values (previous user selections). If required, the users can click over the stack to select among the suitable alternatives. Please observe that the initial and last datatypes can also be discovered by jORCA through the Magallanes integration. In the current version, only the tools from the same repository can be used to create simple workflows.

jORCA logs all the user activity locally and allows users to re-launch any task previously invoked, including chains of tasks as a manually configured workflow. A natural next step in the development of jORCA was the ability to create personalized

pipelines from a set of selected (pre-invoked) operations (found in the log). Once the pipelines have been defined, jORCA is able to enact the full pipeline of services providing updated service invocation status. Finally, by embedding Magallanes in jORCA we close the circle, by discovery, composing, storing, enacting and monitoring pipelines that can also be exported to a Taverna compatible workflow (as SCUFL) or saved in jORCA's list of favorites for easy re-enactment.

Table 1. Current repositories integrated by jORCA showing the number of registered services. The high number of Web Services makes the task of discovering the right service for a specific type of data difficult.

Institution	URL	Repository	#Services
Calgary University	http://www.ucalgary.ca	Moby Canadá	1563
International Rice Research Institute	http://cropwiki.irri.org	IRRI Philippines	1805
Instituto Nacional de Bioinformática	http://www.inab.org	INB stable	150
Advancing Clinico Genomic Trials on Cancer	http://www.eu-acgt.org	ACGT Stable	192
DNA Data Bank of Japan	http://www.ddbj.nig.ac.jp	WABI	21
European Bioinformatics Institute	http://www.ebi.ac.uk	EBI	34

3 Results

In Section 2, we describe the new functionality of jORCA with regards to service composition and enactment of resulting workflows. In order to better illustrate the new functionality, we will here describe a typical user exercise. Note that a more detailed description of the exercise is available as Supplementary Material at jORCA's web page, together with comprehensive material for training.

In the exercise, we will reproduce the workflow proposed in [11] which starts with clustering of co-regulated genes and produces a hierarchical multiple sequence alignment based on similarity of promoter configurations. To conduct this exercise the user must use Magallanes plugin to compose services (i.e. create a pipeline) between FASTA_NA_Multi data-type (set of co-regulated genes in FASTA Format) to Newick_Text (tree that represents the desired multiple sequence alignment), jORCA will then display the different minimal pipelines between both data-types. It is possible that several options are discovered. In this case the user should select the best option. Once finished, the pipeline can be exported (as a Taverna compatible workflow) or directly enacted within jORCA.

Specification of Parameters and Workflow Enactment

When the user decides to enact a pipeline, jORCA requests the required parameters. By default, only parameters required by the first service in the pipeline and suggested

names of the outputs are displayed by jORCA. All other services use the default parameters (if any). All the drag and drop and data-conversion features from jORCA are also available for pipeline enactment. Secondary (non-mandatory) parameters are initially hidden to avoid complex interfaces (see Figure 2), but the user can configure these secondary parameters, for instance distance parameter of the SOTA Clustering algorithm used in the example or whatever other parameter for intermediate tools. Colour codes are used by jORCA to warn the user about mandatory inputs that must be filled. It is also common that a tool output could match with more than one input of the next tool in the execution chain. In this situation, the software requests guidance from the user.

Once enacted, jORCA takes the advantage of its tool tracking to display the progress status by colouring the different tools in the graph (see Figure 2). jORCA also allows saving the results of intermediate tools in the user file system which are then possible to browse or visualize through jORCA.

Once the enactment has finished, the user can visualize the full results (or intermediate results). In the example a Newick tree viewer [9, 25] is used to illustrate this feature. Step by step enactment of this example is available at jORCA's web page.

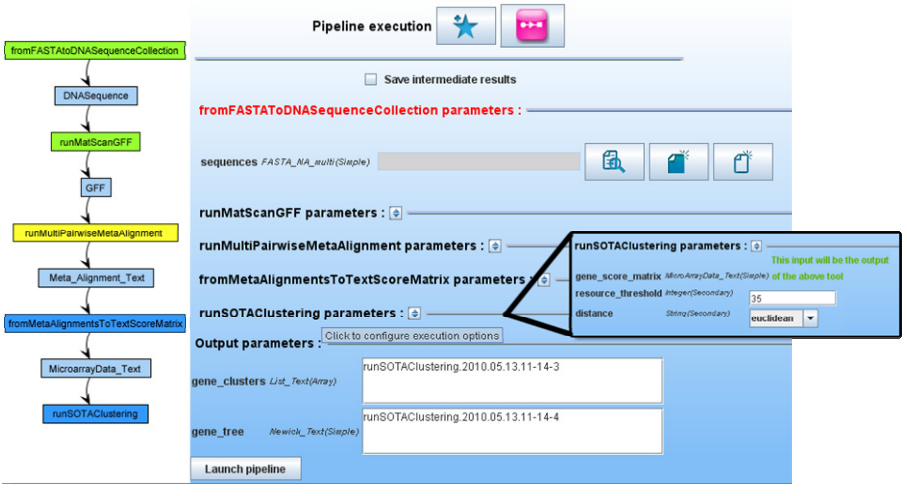


Fig. 2. jORCA pipeline execution main view. On the left, the pipeline execution status graph (Tools will be displayed in green for successfully invoked, yellow running and red failed), on the right, the parameter window with the different parameters of the intermediate services hidden (they can be displayed by clicking the arrow buttons). By default, only the parameters of the first tool and the outputs of the pipeline are displayed. Detailed parameter configuration is displayed for SOTA Clustering. On the top, the options to add the pipeline to the list of favorites and to export the pipeline to Taverna are shown.

4 Discussion

In comparison with the approach of this paper, many studies describe relatively small service catalogues, limiting their practical utility. Approaches using DAML-S, OWL-S

or SAWSDL (see for example [2, 24]) shows the potential utility of well-defined semantic descriptions using established formats for sharing semantic information on the Internet. Service developer participation in the semantic descriptions of services is essential but getting such active participation is difficult in practice. This conclusion is supported by [23] who expresses concerns about the feasibility of large-scale automated workflow composition. Authors claim that, in the domain of bioinformatics, researchers are reluctant to give up control over their experimental setups. In particular, shim services (used to transform data between different data formats and data types) are expected to be the most suitable services to be annotated semantically.

The issue of participation from service providers in annotation of services with semantics is complex although, in particular, end-user participation in annotation is essential. There have been some attempts at improving service annotations through community activities ("jamborees"). These events improve the quality of the annotations but initiatives should be made by the bioinformatics organizations themselves to arrange such events much more often.

The strategy in [4] is to simplify interactive service composition of BioMOBY services, displaying in each step of the workflow construction process only those services that are compatible and more likely to be useful by ranking the services according to several aspects such as semantic similarity of data type inputs or non-functional measurements such as number of retrievals of service definitions from MobyCentral. jORCA performs automatic service composition, generating the entire workflow from start to finish (with the limitation that a single repository can be used at once). After the workflow has been generated, the user can select alternative paths. This strategy is less interactive than [4] but jORCA was not intended to be a complete workflow editor. Instead, focus in jORCA has been placed on service, data type and workflow discovery. Service and data type discovery allows clients to find the required data types and services using text searches in descriptions. Workflow discovery is supported both in the obvious way, by treating workflows as services but also by generating interesting workflows on-the-fly, allowing users to "discover" potentially interesting workflows and then export to a fully-fledged workflow editor such as Taverna.

The issue of locating data types as input for workflow generation approaches is not addressed in [4]. This problem becomes serious considering the massive number of available data types in BioMOBY (currently over 800 data types). jORCA lets the user select a data type from the results of the initial discovery process as source or target for the workflow generation algorithm.

The BioMOBY extension to the Taverna workflow software is reported in [10]. Naturally, the scope and the intended use of that extension is different from jOrca and it is therefore not easy to compare. While jOrca supports the workflow format of Taverna for workflow representation and storage and will be able in future versions to enact Taverna workflows, it is not otherwise tied or integrated with the main Taverna application. The following extensions to Taverna with regards to BioMOBY are reported in [10]:

- Object constructors/deconstructors
- Secondary parameters
- Semantically-aided workflow design
- Enhanced support for collection inputs and outputs

Additionally, jORCA also has the following functionality.

- Multiple repositories (not only those with BioMOBY services) can be added and used in the tool invocation and workflow creation.
- Advanced discovery methods (Magallanes plugin) which can be used to find the correct tool, find the initial and final data-types, create workflows and easily enact them.
- A user can execute a set of tools, and once he/she has found the correct invocation pipeline he/she can create a personalized workflow thanks to the drag and drop functionality.

jORCA attempts to provide a simple approach to web service usage and attempts to avoid creating a complex workflow composition tool. All jORCA features for usability can be used to create, execute and compose workflows.

In summary, the BioMOBY extension to Taverna is tightly integrated with Taverna and is intended directly to facilitate construction of BioMOBY workflows. jORCA is a stand-alone application intended to facilitate, in general, the use of (individual) services and now also workflows. Naturally, the workflow editing functionality provided through Taverna and the BioMOBY extension is much more advanced. In this sense, jOrca and the BioMOBY extension of Taverna complement each other: basic workflows (pipelines) can be created with jORCA and then exported to Taverna where advanced editing is possible thanks to the BioMOBY extension. It is also worthwhile to note that while the examples in this paper have been for BioMOBY services and Taverna workflows, jORCA is not limited to those technologies.

Combining Different Service Standards in a Workflow

BioMOBY services adhere to a predictable standard for data representation through the use of shared datatype ontology. Therefore, it is possible to deduce the (syntactic) and to some extent, semantic compatibility between the services (provided that the services use sensible datatypes which represent the semantics of the inputs and outputs).

The automatic workflow composition is only possible in such cases. However, a more “manual” approach of service composition is conceivable. If the user mixes BioMOBY and non-BioMOBY services (such as a more traditional WSDL-described service), the question of datatype mapping becomes important.

Due to the Modular API, which jORCA is based on, it is possible to create specific formatters and loaders. The Modular API supports an underlying data representation model (in these examples, this is directly mapped to the BioMOBY data representation ontology. If it is necessary to combine services with different protocols (for example BioMOBY and traditional WSDL-described SOAP services), the formatter and an intelligent heuristic can translate data between the models, acting as a kind of data broker. This works for simple datatypes (String, Integer etc) but for more complex datatypes it would be necessary to programmatically create a loader. The loader is able to do more complex transformations to add additional information as prescribed by an individual data model (for example, sequences in BioMOBY contain not only the sequence itself but also the length: this can be calculated on-the-fly by a loader). Naturally, this is not an end-user task since developing those formatters and loaders

require some programming ability. However, several standard formatters and loaders are available out-of-the-box and more can be added and configured by the software developer. The approach is limited as there are very many different datatypes in bioinformatics but, at least, works well for some basic datatypes. Advanced object creation methods have been added to jORCA as reported in [12], integrating BioData-SF library and plug-ins, intelligent wizards that use heuristics have been integrated to facilitate Web Service composition and object creation. Inter-protocol workflow creation has not been implemented yet, but this feature is considered for future releases. An example of manual inter-protocol Web Service combination can be found in jORCA's web page (Guided exercise "Using Web Services - different protocols and repositories"). Nevertheless, BioMOBY services are easy to combine and, for end-users, we recommend to create workflows with only BioMOBY services.

Service Metadata Repository

Our approach is able to directly work over several existing sets of bioinformatics web-services by the development of specific access components. Currently, components exist to connect Magallanes to BioMOBY service registries and a registry for grid-services (ACGT project [19]). In this paper, we have focused on using the BioMOBY service registry, which has an active developer community and almost 1600 available bioinformatics web-services (beginning of 2009).

This has the advantage that Magallanes is able to work with a complete catalogue of bioinformatics services, both for service discovery and for workflow generation. In particular, the BioMOBY approach with a shared data type ontology results in services that are, in principle, easy to connect since they do not require intermediate data formatting to be compatible.

However, the disadvantage of using the BioMOBY registry (MobyCentral) as a source for service metadata is that, while the number of services is high, metadata is supplied directly by the service providers without any quality control. For example, MobyCentral has almost 800 data types (January 2009). This could mean problems in service integration when services that really consume similar data are not considered as compatible with a given data because the services were incorrectly annotated with different data types.

The conclusion is that data type ontologies must be large enough to express the required data types but should also be controlled to ensure that service providers do not unnecessarily define new data types when a suitable data type already exists. This is essential to enable meaningful service composition. The approach to data type ontology curation adopted by the Spanish National Institute for Bioinformatics (INB) is to maintain a data type ontology which is compatible with the ontology from BioMOBY but dedicate special effort to promoting service compatibility. New services and data types are suggested by service providers but are not added to the official INB service catalogue without prior approval from an ontology committee. A development catalogue where new data types and services can be registered without restriction is available for testing. This approach is less scalable than the end-user oriented approach in MobyCentral but can result in a good trade-off between openness (scalability) and control.

Regarding the scalability of our approach, jORCA uses the Java regular expressions package to perform searches. In order to improve the efficiency, Magallanes optimizes and compiles the pattern before start to search. As a result, the search time increases linearly with the search space and with the number of keywords if it is an 'OR' search.

5 Conclusions

The profusion and dispersion of bioinformatics tools make routine tasks of data processing complicated. Users are involved in the tedious and error-prone task of copying and pasting intermediate results, or adjusting data from a particular output for further processing with another tool. Under this scenario, workflows are becoming the new paradigm for repetitive and complex data processing in bioinformatics.

In previous communications, we reported two different applications (jORCA and Magallanes) to facilitate the discovering, interconnection and exploitation of bioinformatics tools. In this work, we present an extended version of jORCA that integrates discovery, composition, storage/exporting, enactment and monitoring of Web Services pipelines. Following the original philosophy of the applications, with usability high on the top of priorities for jORCA, the new features take into account the user comfort and the existing look and feel of jORCA.

The strength of software integration is demonstrated (details in the Complementary Material) by reproducing the elaboration and enactment of a published workflow, spending few minutes to complete the task, including monitoring of enactment and visualization of results.

In the agenda for enhancing the workflow management side of jORCA, we are planning functionality to suspend and resume workflow enactment, complex (multiple branches) workflow enactment and more complete workflow edition (for example, that allows calling to local procedures or programs).

Our work has so far focused in using the ontologies maintained by INB and BioMOBY. However, it would be interesting to consider other ontologies such as EDAM [5] (used in the Embrace Registry [15]) or the service ontology from MyGrid [23] (the latter is very similar to the BioMOBY ontology). jORCA is based on an extendible framework (see section 2) and a new access module could be written for those ontologies. It is worthwhile to note that this would not involve large (if any) modifications to the main application (jORCA).

Another limitation of this work is that as full workflow composition hasn't been implemented. This will be a future extension of these new features, creating workflows with several branches.

jORCA is available under the Creative Commons Attribution - No derivative works 2.5 license and the source code (entirely implemented in Java) is available upon request. Download the latest version, documentation, guided exercises and videos from jORCA's web. In that page you can also get informed of the new features through jORCA's mailing list.

Acknowledgements

This work has been partially financed by the National Institute for Bioinformatics (www.inab.org) a platform of Genoma-España; the EU project "Advancing Clinico-Genomic Trials on Cancer" (EU-contract no.026996) and the RIRAAF Spanish network on allergies (RD07/0064/0017).

Author would like to thank the BITLAB research group for their invaluable help in software benchmarking. A special thanks to Maximiliano García and Antonio Muñoz for the rigorous and continuous user feedback.

References

1. BioMoby Dashboard, http://biomoby.open-bio.org/CVS_CONTENT/moby-live/Java/docs/Dashboard.html (accessed 2010-07-08)
2. Cardoso, J., Sheth, A.: Semantic e-workflow composition. Journal of Intelligent Information Systems 21(3), 191–225 (2003)
3. Carrere, S., Gouzy, J.: Remora: a pilot in the ocean of BioMOBY web-services. Bioinformatics 22, 900–901 (2006)
4. DiBernardo, M., Pottinger, R., Wilkinson, M.: Semi-automatic web service composition for the life sciences using the BioMoby semantic web framework. Journal of Biomedical Informatics 41, 837–847 (2008)
5. EDAM ontology, http://edamontology.sourceforge.net/ (accessed 2010-08-04)
6. Goble, C., et al.: BioCatalogue: A Curated Web Service Registry for the Life Science Community; Communication Data Integration in Life Sciences (DILS) 2009 - ENFIN Collocated Workshop, Manchester, UK (2009)
7. Gordon, P.M.K., Sensen, C.W.: Seahawk: Moving Beyond HTML in Web-based Bioinformatics Analysis. BMC Bioinformatics 8, 208 (2007)
8. Gordon, P., Sensen, C.: A Pilot Study into the Usability of a Scientific Workflow Construction Tool (2007), http://hdl.handle.net/1880/46486
9. Han, M.V., Zmasek, C.M.: phyloXML: XML for evolutionary biology and comparative genomics. BMC Bioinformatics 10, 356 (2009)
10. Kawas, E., et al.: BioMoby extensions to the Taverna workflow management and enactment software. BMC Bioinformatics 7, 523 (2006)
11. Kerhornou, A., Guigó, R.: BioMoby Web Services to support clustering of co-regulated genes based on similarity of promoter configurations. Bioinformatics 23(14), 1831–1833 (2007)
12. Martín-Requena, V., Ríos, J., García, M., Ramírez, S., Trelles, O.: jORCA: easily inte grating bioinformatics Web Services. Bioinformatics 26(4), 553–559 (2010)
13. Navas-Delgado, I., et al.: Intelligent client for integrating bioinformatics services. Bioinformatics 22, 106–111 (2006)
14. Oinn, T., et al.: Taverna: a tool for the composition and enactment of bioinformatics workflows. Bioinformatics 20, 3045–3054 (2004)
15. Pettifer, S., et al.: An active registry for bioinformatics web services. Bioinformatics 25, 2090–2091 (2009)
16. Ramirez, S., et al.: A flexible framework for the design of knowledge-discovery clients. In: International Conference on Telecommunications and Multimedia (2008)

17. Ríos, J., Karlsson, J., Trelles, O.: Magallanes: a Web Services discovery and automatic workflow composition tool. BMC Bioinformatics 10, 334 (2009)
18. Romano, P., et al.: Biowep: a workflow enactment portal for bioinformatics applications. BMC Bioinformatics 8(suppl. 1), s19 (2007)
19. Tsiknakis, M., et al.: Building a European biomedical grid on cancer: the ACGT Integrated Project. Studies in health technology and informatics 120, 247 (2006)
20. Web services architecture working group, `http://www.w3.org/2002/ws/arch/` (September 2006)
21. Wilkinson, M.D., et al.: Interoperability with Moby 1.0–it's better than sharing your toothbrush! Briefing in Bioinformatics 9(3), 220–231 (2008)
22. Wilkinson, M.D.: Gbrowse moby: a web-based browser for BioMOBY services. Source Code for Biology and Medicine 1, 4 (2006)
23. Wolstencroft, K., et al.: The myGrid ontology: bioinformatics service discovery. Int. J. Bioinformatics Research and Applications 3(3), 303–325 (2007)
24. Wu, Z., et al.: Automatic composition of semantic web services using process mediation. In: Proceedings of the 9th Intl. Conf. on Enterprise Information Systems ICES (2007)
25. Zmasek, C.M., Eddy, S.R.: ATV: display and manipulation of annotated phylogenetic trees. Bioinformatics 17, 383–384 (2001)

Supplementary Material

The full description of the use case from the Results section is available at http://www.bitlab-es.com/jorca.

A Linked Data Approach to Sharing Workflows and Workflow Results

Marco Roos[1,2], Sean Bechhofer[3], Jun Zhao[4], Paolo Missier[3], David R. Newman[5], David De Roure[6], and M. Scott Marshall[2,7]

[1] BioSemantics Group, Department of Human and Clinical Genetics, Leiden University Medical Centre, P.O. Box 9600, 2300 RC Leiden, The Netherlands
[2] Informatics Institute, Faculty of Science, University of Amsterdam, P.O. Box 94323, 1090 GH Amsterdam, The Netherlands
m.roos@lumc.nl
[3] School of Computer Science, The University of Manchester, Manchester, UK
{paolo.missier,sean.bechhofer}@manchester.ac.uk
[4] Jun Zhao, Department of Zoology, University of Oxford, South Parks Road, Oxford, OX1 3PS
jun.zhao@zoo.ox.ac.uk
[5] School of Electronics and Computer Science, University of Southampton, Highfield Campus, University Road, Southampton SO17 1BJ, UK
drn05r@ecs.soton.ac.uk
[6] Oxford e-Research Centre, University of Oxford, 7 Keble Road, Oxford OX1 3QG, UK
david.deroure@oerc.ox.ac.uk
[7] Department of Medical Statistics and Bioinformatics, Leiden University Medical Centre, P.O. Box 9600, 2300 RC Leiden, The Netherlands
mscottmarshall@gmail.com

Abstract. A bioinformatics analysis pipeline is often highly elaborate, due to the inherent complexity of biological systems and the variety and size of datasets. A digital equivalent of the 'Materials and Methods' section in wet laboratory publications would be highly beneficial to bioinformatics, for evaluating evidence and examining data across related experiments, while introducing the potential to find associated resources and integrate them as data and services. We present initial steps towards preserving bioinformatics 'materials and methods' by exploiting the workflow paradigm for capturing the design of a data analysis pipeline, and RDF to link the workflow, its component services, runtime provenance, and a personalized biological interpretation of the results. An example shows the reproduction of the unique graph of an analysis procedure, its results, provenance, and personal interpretation of a text mining experiment. It links data from Taverna, myExperiment.org, BioCatalogue.org, and ConceptWiki.org. The approach is relatively 'light-weight' and unobtrusive to bioinformatics users.

Keywords: Linked Data, Semantic Web, Digital preservation, Workflow, Provenance, Concept Web.

T. Margaria and B. Steffen (Eds.): ISoLA 2010, Part I, LNCS 6415, pp. 340–354, 2010.

1 Introduction

A commonly used approach to the study of biological systems in the omics era is to integrate information from multiple resources, often in the context of interpreting our own data from an in-house omics experiment (e.g. genome-wide gene expression). The bioinformatics analysis pipeline is therefore usually complex, while the amount of relevant knowledge that could theoretically be considered for a new hypothesis is daunting. With over 19 million biomedical publications in PubMed alone and over a thousand public databases, information overload is a general problem in biology. Although these numbers are impressive, the abundance of information only translates into knowledge gain if we can locate and leverage the knowledge contained in the many distributed resources, including derived data and knowledge extracted by workflows or other computational means. Many have responded to the challenge by aggregating valuable or otherwise thematic data in data warehouses and making the integrated data available on the Web in the form of a knowledge base. However, in all cases, the challenge remains to create a system for the description and subsequent computational discovery of distributed knowledge resources so that they can be incorporated into additional experiments and hypothesis testing.

Not surprisingly, sharing a bioinformatics experiment and its results can be challenging, whether for reuse of its results and its methodology or for peer evaluation. In a networked environment, sharing involves a search process in order to select from a potentially vast number of varied offerings. For wet laboratory experiments, this is supported in particular by the 'Materials and Methods' section of a publication, which describes how the results were obtained. It describes the protocol that was followed, often referring to protocols in earlier publications or in journals and books dedicated to protocols. It describes the specimens and equipment used in enough detail to reproduce the experiment. In many cases, strict nomenclature is imposed by publishers to name, for example, genes and organisms. This makes it easier for peers to understand the experiment, thereby facilitating its review and reuse of its protocol. The Materials and Methods section is considered one of the pillars of experimental biology and probably the most critically assessed part of scientific discourse. A digital version of the Materials and Methods for a bioinformatics experiment would increase its reusability, as well as the rigor by which it can be evaluated. However, a digital equivalent of the 'Materials and Methods' section does not yet exist in bioinformatics. In this paper, we show how a workflow system and web-based information repositories can be used to create the digital equivalent of the Materials and Methods section when we adopt a Semantic Linked Data approach. In the remainder of this paper, we describe the user requirements and their technical counterparts, before describing the components that provide the basis of our approach, supported by a proof of principle in section 3.

1.1 Motivating Scenario

To motivate our approach with a scenario we introduce Alice. Alice is interested in performing a bioinformatics experiment to discover proteins that interact with transmembrane proteins, particularly those that can be related somehow to neurodegenerative diseases in which protein aggregates (amyloids) play a significant role (e.g.

Huntington's Disease and Alzheimer's Disease). Alice would like to reinvent as little as possible, thus reuse any previously developed analysis pipeline that she can trust to be of the appropriate relevance and quality. Consequently, the typical experiment cycle may contain these four steps:

(i) *Retrieve:* Alice needs to find a previously published analysis pipeline that will suit her needs, and retrieve all the relevant resources (data and methods) for her own analysis.
(ii) *Review:* she will want to review the analysis pipeline before she uses it and study the evidence that led to the interpretation of the data that it previously produced. In theory, the aggregation of metadata associated with the previous experiment should suffice to completely understand the process from input to output to biological interpretation.
(iii) *Repeat, Reuse, Repurpose:* first, Alice would like to repeat a previous analysis to evaluate the process step by step as part of reviewing and validating it. Secondly, Alice would like to be able to run (parts of) an analysis pipeline again for her own purposes, much like bench biologists design new experiments from previously published protocols.
(iv) *Conserve:* when Alice has performed her own analysis, she would like to conserve her design and the association of the analysis with her results, her interpretation, and her initial hypothesis. A bench biologist would keep this type of information in a laboratory journal as the basis for a publication. Alice would like to keep notes on (intermediate) results, the steps that she performed at a particular time, the protocols she used, and any additional information that she may need to support her interpretation of the data. Obviously, the quality of this step determines how effectively Alice's colleague, Bob, would be able to evaluate and reuse Alice's work.

In the next session, we discuss bottlenecks and requirements for performing these steps effectively for bioinformatics experiments.

1.2 Bottlenecks for Evaluating a Bioinformatics Experiment

We identify the following bottlenecks for biologists who wish to be able to evaluate and reuse a bioinformatics experiment:

1. **Retrieve.** Currently, search engines such as Google and NCBI's PubMed are the most common tools to find related work, including methods and (references to) data. This may serve some purposes well enough, but is limited by how precisely we can formulate a search query. In the scenario above Alice will find that it is difficult to find a protein interaction discovery method in the literature using these tools. Most titles refer to a biological finding rather than the method that was used. She will find that data can often be retrieved on request from the authors or via a public database, but the original analysis pipeline is often not readily available, nor its component parts. Alice will often find it frustrating that her desired method cannot be used independently from the monolithic application in which it is embedded. In this paper, we refer to workflows in myExperiment and Web Services in BioCatalogue to address these issues. Other partial solutions have been developed for bioinformatics, such as BioConductor, popular for developing and

sharing statistical analysis methods in the R language [1], or BioMoby, a project that pioneered the use of semantically annotated web services [2].

2. **Review.** While Alice may read the authors' description of a bioinformatics experiment in a publication, she will often find it hard to evaluate its steps. She will not be able to obtain an evidence graph from input to output to biological interpretation, i.e. no data provenance that links between the analysis pipeline and its results is available. The feature-rich (web) application mentioned above is not sufficient to evaluate the underlying computational pipeline. Moreover, additional information that Alice would like to use for her evaluation can be hard to access. For instance, she may want to find which parts in a pipeline were based on other pipelines, which scientists corroborate previous results, or which diseases are associated with the proteins in the result set. There is currently no standardized interface in bioinformatics that makes it possible to query across data, methods and interpretations. In this paper we demonstrate the use of Linked Data and RDF (See Section 1.3), but these are not yet commonly applied in this context.
3. **Repeat, Reuse, Repurpose.** It is often difficult to repeat a Bioinformatics experiment. As mentioned, component parts may not be available for a new application and even when a client application is available to rerun the full pipeline, the underlying databases may have been updated or computational methods improved. This cannot be completely controlled when applications are built on 3rd party resources, but Alice would be helped greatly if she was notified of changes such that she can take these into account when rerunning a method. Workflows built from Web Services seem address part of these bottlenecks.
4. **Conservation.** In the laboratory, the most generally accepted method for conservation of methods, data, and interpretation is still traditionally through publications and lab journals. Many publishers offer the option of supplying additional digital information, but the quality of this 'supplementary information' varies and it is not usually computationally accessible due to a lack of standardization, nor does it provide a way to link the analysis pipeline, its results, and associated metadata. For example, although it is common practice to upload raw data from microarray studies to ArrayExpress and GEO, the lists of differentially expressed genes commonly referred to by articles are not disclosed with the raw data. However, it is precisely these lists that are the subject of discussion in any associated articles. Alice will find that she has no direct way to associate her notes (her annotations) with her analysis design and its results.

New Web2.0-inspired applications provide alternative ways to digitally conserve analysis designs (myExperiment; [3]), their component parts (BioCatalogue; [4]), and the concepts used in biological hypotheses and personal notes (ConceptWiki[1]; [5][2]). In this paper, we describe how we use RDF to link some of these resources together to create a comprehensive digital resource that describes the 'materials and methods' of a bioinformatics experiment, and we discuss how this addresses Alice's bottlenecks. First, we identify these additional user requirements:

[1] http://ConceptWiki.org
[2] Originally based on WikiProfessional technology: http://wikiprofessional.org

1. **Comprehensive.** Alice would be helped in reviewing a previous analysis if she would be able to query a comprehensive 'warehouse' of information about the methods and the data associated with an experiment. For instance, she may want to look at alternative gene names, related diseases, or author names and affiliations.
2. **Light Weight.** While Alice would like to query a comprehensive 'warehouse' to perform an extensive review, she would not want to spend substantial effort to build this warehouse herself. Moreover, she would not want to do so for her own analysis that Bob will review.
3. **Transparent.** The technology for digital conservation relies on semantic annotation of the components of an experiment and its results. However, this should not interfere with the design of the bioinformatics analysis. In fact, Alice should feel supported by it, for instance by implementing it as a tool that helps her keep a laboratory journal. The activities that result in a digital version of materials and methods should ideally be part of her routine research activity.
4. **Personal.** In general, reusing a community consensus model to annotate the results of an analysis will help Alice and Bob to interpret her results. However, Alice's work is cutting-edge, so she has a personal view of her bioinformatics experiment that is reflected in her hypothesis and data interpretation. Therefore, Alice requires the ability to use the most appropriate model for her annotations, and the ability to extend an existing model with concepts that she is missing.
5. **Shared terminology, Identity and Reference.** In biological discourse, various 'nomenclatures' (e.g. for species or gene names) are used to resolve ambiguity. Also for a bioinformatics analysis we depend on unambiguous and unique identifiers for the objects in our digital materials and methods. In this paper, we use the Concept Web, a new part of the Semantic Web that aims to be a world-wide resource of disambiguated (biological) concepts, machine readable through RDF and identified by universally unique identifiers.

1.3 Semantic Web, RDF and Linked Data

The Semantic Web as described by W3C[3] is about providing common formats for integration and combination of data drawn from diverse sources. The Semantic Web aims to lift us from a web of pages or resources with data intended solely for human consumption to a "web of data", with this data explicitly exposed, rather than locked away inside particular applications.

The Resource Description Framework (RDF[4]) is seen as a key technology in the publication of the web of data, including data from the Life Sciences [6, 7]. RDF provides a common triple-based data model for publication of data. It is indeed increasingly used to expose data sets and resources as RDF graphs. SPARQL[5] provides a language for querying graph patterns within RDF graphs, and also defines a protocol that describes how queries can be conveyed to a SPARQL "endpoint", a service that processes SPARQL queries. SPARQL thus enables the query of RDF data sets and provides a common infrastructure on which to build applications.

---

[3] http://www.w3.org/2001/sw/
[4] http://www.w3.org/RDF/
[5] http://www.w3.org/TR/rdf-sparql-query/

An approach that is steadily growing in popularity is that of Linked Data [8, 9]. Linked Data is a set of guidelines or best practices that have been introduced in order to facilitate the exposure and connection of different data sets. The Linked Data approach relies heavily on RDF and the use of URIs to identify objects or concepts that are being described. Linked Data advocates the following principles:

1. Use URIs to identify objects/concepts, in particular use HTTP URIs which are then dereferencable.
2. Provide useful information when those URIs are dereferenced, ideally using standard formats and representations (e.g. RDF)
3. Provide links to other URIs, so that applications can discover more.

The adoption of these guidelines for the publication of data enables the integration of data sets from a wide range of domains, with significant efforts in the life sciences. Key issues facing the Linked Data approach include the provision of common, shared identifiers for the objects that are being described -- the use of common URIs drives the "linking" in Linked Data. Ensuring that applications and datasets use common identifiers is thus crucial in facilitating this linking. Initiatives such as Shared Names[6], Okkam[7] and the Concept Web[8] (as discussed later) are aiming to provide URIs for publicly available records. Authoritative resources such as UniProt, PubMed and EntrezGene are also being exposed as RDF via SPARQL endpoints by projects including Linked Life Data[9] or Bio2RDF[10].

In the context of our scenario, there are additional objects that can be identified and linked together, as discussed below. These include the *workflows* that are used to process the data, the *services* that are used within those workflows, the *researchers* who conduct the research and the *outputs* (papers, presentations etc) that those researchers produce. Exposing all of these resources as Linked Data will provide a rich, connected space facilitating discovery, analysis and reuse of digital materials and methods.

2 Resources for Digital Materials and Methods

Here we describe how the Linked Data principles are used to aggregate the resources that Alice could use for (i) retrieving a previously constructed pipeline for protein discovery, its component parts, and associated documentation (myExperiment, BioCatalogue), (ii) reviewing the analysis and its results (Taverna workflow provenance with domain specific extensions), (iii) repeating the analysis for her own purposes, and (iv) classifying the results: protein interactions found by a text mining workflow (Taverna+AIDA plugin). Only a limited number of additional links are necessary to create a new aggregation that represents the digital materials and methods of Alice's experiment. We demonstrate this with an example in section 3.

---

[6] http://sharedname.org/
[7] http://www.okkam.org/
[8] http://www.conceptweballiance.org/
[9] http://linkedlifedata.com/
[10] http://bio2rdf.org/

2.1 RDF: The Model for Linked Data and Comprehensive, Yet Light-Weight Coverage of Experiment-Related Data

Our framework of choice for digitally conserving a computational analysis encompassing hypothesis, provenance, workflow(s), services, data, and interpretation is based on RDF (section 1.3). Many applications have started exposing their data on the web via RDF, making their resources part of the Linked Open Data cloud. This can be done either by a SPARQL endpoint or by providing RDF as a machine readable alternative to the data presented on a web page. This includes the resources that we have identified as useful sources for our digital Materials and Methods: Taverna, myExperiment, BioCatalogue, and the Concept Web. With a minimal number of links between these sources, Alice is provided with a comprehensive amount of metadata about an experiment.

2.2 myExperiment and BioCatalogue: Repositories for Digital Protocols and Their Components

While the workflow paradigm provides a useful way to formalise an analysis pipeline, myExperiment.org provides a repository to share and publish these artefacts on the Web [10]. Additional documentation (tags, comments) can be provided by the owner of a workflow or users of myExperiment. This facilitates their discovery and reuse. In turn, BioCatalogue provides a registry for the components of a workflow, in particular Web Services [4]. Similar to myExperiment, BioCatalogue enables registered users to provide documentation and tag contents, again facilitating their discovery. Both resources provide a REST API and URLs for every object that they contain. Conequently, myExperiment and BioCatalogue are sources of identifiers for use in bioinformatics publications. Versioning and attribution features ensure that specific adaptations of a workflow can be referenced. Attribution allows Bob to link to Alice's workflow and acknowledge her. When Alice also attributes a workflow, then these links implicitly create a chain of references to the origins of a workflow. Finally, we mention myExperiment 'packs': aggregations of (references for) resources both inside and outside of myExperiment. This makes myExperiment a provider of persistent and structured supplemental information. How can we use myExperiment and BioCatalogue to link to Alice's experimental results and create the digital version of her Materials and Methods? MyExperiment also exposes its content as RDF [11]. The motivation is to make the content of myExperiment part of Linked Data, allow it to be linked to other resources and be queried via a SPARQL endpoint. This will allow Alice to retrieve information from the Web of Data starting from a myExperiment pack. The semantic model that was used for myExperiment supports its core features. It represents the social model behind myExperiment and the model that facilitates the management and sharing of workflows and associated components for other users. This 'e-Research' model is extensible such that it can be linked to additional domain specific models. The most straightforward part of the myExperiment semantic model is the representation of the myExperiment mySQL schema in OWL DL. The Simple Network Access Rights Management (SNARM[11]) ontology was used to capture the sharing model of myExperiment. For representing the social content of myExperiment

[11] http://rdf.myexperiment.org/ontologies/snarm/

several ontologies were reused: Dublin Core[12], Friend of a Friend[13], Semantically Inter-linked Online Communities (SIOC[14]), and the Open Archives Initiative's Object Reuse and Exchange ontologies/schemata (OAI/ORE[15]). These shared ontologies facilitate co-reference resolution, which is one of the major tasks on the Semantic Web. It makes it easier to understand the purpose of the classes and relations and facilitates access to semantic data outside of myExperiment. The users of the user interface are never confronted with the full extent of these ontologies. Exposing the content of myExperiment as Linked Data on the web allows Alice to define SPARQL queries for typical Materials and Methods questions such as 'Who did what and when?', or 'Whose work was this workflow based on?'. Moreover, the relatively straightforward action for Alice to upload and publish her workflow on myExperiment provides Bob, a potential new user, additional metadata to investigate. At the time of writing BioCatalogue does not yet expose its content as RDF. For our example in section 3 we used myExperiment RDF as a template to create a mock version of BioCatalogue RDF.

2.3 Workflow and Provenance

Workflows are the most common type of object that Alice finds on myExperiment for reusing in her own work. Workflows are formal and executable models of computational protocols for data analysis experiments. Alice can review the design of a workflow, similar to how she would evaluate a protocol from a laboratory manual. However, the best way to review an experiment before using it for one's own purposes is to evaluate the results that it produced step by step and the personal annotations that the first user of the workflow provided while he/she was running it. In comparison, if Alice was to reuse a wet laboratory protocol by bench biologist Chris, then his laboratory notes made while he was performing the protocol would be the most valuable. First, they contain what was actually done in relation to the results at a particular point in time. Secondly, it contains Chris' personal annotations on how the results should be interpreted. Therefore, capturing a detailed trace, the *provenance*, of each workflow execution (a "run") linked with personal annotations represents a step forward in the direction of recording materials and methods in machine processable form. The Taverna workflow system persistently stores the provenance of workflow runs (for example, the execution of Alice's experiment) and makes it available to scientists for evaluation. At any later time Bob can query and analyse Alice's results. Taverna adopts a semantic data model to represent provenance. The model is specified as an OWL ontology, called *Janu*s. Provenance traces are RDF graphs [12]. The concepts in Janus describe workflow tasks as well as the data that they consume and produce, while the provenance graph captures the actual tasks and the data transformations that they produced during a workflow run.

The choice of a semantic model is designed to facilitate the semantic annotation of provenance graphs with domain-specific concepts, such as those found on the Concept Wiki. When provenance is first recorded, the provenance graphs are

[12] http://dublincore.org/
[13] http://www.foaf-project.org/
[14] http://sioc-project.org/
[15] http://www.openarchives.org/ore/

"domain-agnostic" and semantics-free, but their grounding in RDF and OWL makes it easy to add annotations whenever they become available, and to integrate with the broader Web of Linked Open Data [13]. Such integration involves mapping data elements in the provenance graph to data that is published elsewhere in the Web of Data, making it possible for queries to seamlessly include conditions on properties of the data that were not explicitly represented in the original graph. Henceforth, without bloating the original provenance produced by the workflow enactor, a comprehensive graph can be obtained via meaningful relations on the Semantic Web.

Janus achieves the required linking by reusing a number of shared ontologies found on the web. Formally, Janus is an extension of Provenir [14], which itself extends concepts from the Basic Formal Ontology (BFO[16]). Provenir is an upper-level reference model for capturing provenance, including concepts such as data, process and agent, and several relations such as for partonomy, precedence, and causality. Janus extends Provenir to include terms from the Life Sciences domain. For example, four ontologies were chosen for case studies in genomics from the almost 200 publicly shared models that are available via the National Centre for Biomedical Ontologies (NCBO; [15]): BioPAX[17], the National Cancer Institute Thesaurus[18], the Foundational Model of Anatomy[19], and the Sequence Ontology[20]. Doing so following Linked Data conventions allow Alice and Bob to ask useful biological questions about interacting biological molecules from KEGG, Reactome, and BioCyc databases [16]. As such, provenance becomes the core of a comprehensive digital resource of materials and methods for biologists to evaluate and reuse.

2.4 Concept Web: Repository for Uniquely Identified Concepts, Their Relations and their Evidence

A new approach to providing common identifiers for important terms in scientific discourse is proposed by the Concept Web Alliance [17]. Inspired by the success of Wikipedia, it 'calls upon a million minds' to create and curate a universal resource of disambiguated concepts and basic relations between them [5]. In line with a wiki approach, scientists can register new concepts and improve the information associated with them. Initial content is supplied by terminology resources such as UMLS, UniProt, and the ontologies that can be obtained from NCBO's bioportal [15]. Relations can be aggregated to form so-called 'nano-publications' [17]. 'Malaria' and 'mosquito' are example concepts, while 'Malaria *is caused by* mosquitos *as discovered by* Charles Laveran *in* 1880' could be a nano-publication including a trace to evidence. Each concept, relation, and nano-publication will have its own universally unique identifier that is persistent and immutable over time. Therefore, Alice and her peers can use Concept Web identifiers as stable references to their data instead of, for instance, gene names, which can change. Because the Concept Web is also part of the Semantic Web and exposes its content as RDF, it is a unique source of identifiers for use on the Web of Data. In our proof of principle, we will use concepts from the

[16] http://www.ifomis.org/bfo
[17] http://www.biopax.org/
[18] http://ncit.nci.nih.gov/
[19] http://sig.biostr.washington.edu/projects/fm/
[20] http://www.sequenceontology.org/

Concept Web as our point of reference for all digital objects except those from myExperiment and BioCatalogue. These resources already claim that their URLs are persistent and universally unique.

3 Proof of Principle

3.1 Linking Experimental Results and Evidence (Taverna Provenance), Personal Interpretation (AIDA Plugin), Digital Protocol (myExperiment) and Its Components (BioCatalogue), in Terms of Biological Concepts (ConceptWiki)

Here we show how we obtain a snapshot out of the digital, machine readable Materials and Methods as a result of Alice running her workflow. Bob would like to use this information to review how Alice obtained these results, and study some additional information about these results. Our example is derived from the workflow that Alice was using for protein discovery. We have used a number of resources that Bob would need to satisfy his information needs. When resources follow the Linked Data principles we require only a minimal number of relations to embed Alice's workflow results in a large network of references. Therefore, this solution is light-weight, but still comprehensive. The following Linked Data resources were used:

1. Taverna provenance: exposed as RDF using *Janus* (section 2.1)
2. myExperiment: a provisional RDF document for the protein discovery workflow was obtained from the myExperiment development server (section 2.2)
3. BioCatalogue: we created a mock RDF document using myExperiment RDF data as example. A RDF interface similar to that of myExperiment is planned (Jiten Bhagat, personal communication)
4. ConceptWiki: provisional RDF documents were obtained from the ConceptWiki development server. We created new concepts via the ConceptWiki interface to obtain universally unique identifiers for the creator of the workflow and services. Ideally, myExperiment and BioCatalogue would use these as identifiers as well.
5. UniProt: the RDF document for our example protein was obtained from the RDF interface of the main UniProt web site.

To link these resources, we used properties from the following ontologies:

1. A Workflow ontology previously created for structuring data from a workflow [18]
2. A mapping ontology for mapping between a (text mining) process and biological results [18]
3. The Semantic Web Applications in Neuromedicine (SWAN; [19]) ontology version 1.2[21]
4. The Relation Ontology (RO) from the OBO Foundry [20]
5. The Dublin Core (DC) meta-thesaurus.

[21] http://swan.mindinformatics.org/ontology.html

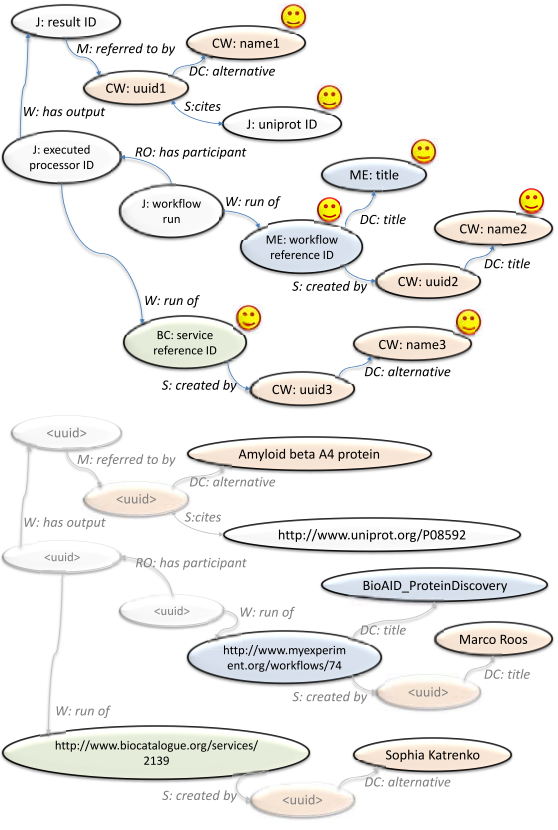


Fig. 1. Top: graphic representation of an evidence query. The smiley symbols indicate which elements could be in the output for human consumption. Bottom: graphic representation of the result of the evidence query. '<uuid>' indicates a universally unique identifier that is provided by any of the resources. J: Janus Provenance Ontology; M: mapping ontology to relate the (text mining) process to biological concepts; CW: Concept Web (the Concept Wiki is the human GUI); S: SWAN 'Semantic Web Applications in Neuromedicine' ontology; W: Workflow ontology; RO: OBO Relations ontology; ME: myExperiment; DC: Dublin Core; BC: BioCatalogue. See the appendix for the SPARQL representation and its results.

The following links were made (see appendix for the commented RDF):

Between Taverna provenance and reference resources:

- a workflow run *is a run of* a workflow on myExperiment
- an executed processor *is a run of* a service on BioCatalogue
- a workflow result *is the result of* a service run
- a workflow result *refers to* a concept on the Concept Web

Between BioCatalogue, myExperiment, Concept Web, and UniProt

- a service in BioCatalogue *is an element of* a workflow on myExperiment
- a workflow *is created by* a user who is identified on the Concept Web
- a service in BioCatalogue *has a creator* who is identified on the Concept Web
- a Concept Web entry *cites* a UniProt entry and vice versa

Missing links

- a processor_exec *participates in* a workflow run

When the results of the Protein Discovery workflow are linked to the resources that were used to create them, and to the resources that refer to and describe her results, Bob can obtain a comprehensive view to review for instance how she obtained her results, who were the people responsible for the methods and the workflow, and the 'linked in' identifiers that Bob could use to further interpret Alice's results. Bob can obtain this information by querying the RDF graph. We demonstrate this by a 'materials and methods' query (figure 1). First we created a SPARQL endpoint by uploading the RDF documents described above to our Sesame RDF repository. For our proof of concept we focussed on one of the workflow results, the protein 'Amyloid beta A4 protein' (UniProt identifier: P08592[22]). The graph pattern in figure 1 shows how evidence for a workflow result was queried (see the appendix for the full SPARQL query). Bob would be able to study this evidence, and continue querying for new information. For instance, Bob could write a query that retrieves all literature citations contained in a UniProt RDF document that was ultimately linked to Alice's workflow result as created by Taverna's provenance engine.

4 Discussion and Conclusion

How do the components that we have presented alleviate the bottlenecks that Alice and Bob face in their research? What are the potentially new bottlenecks that we have not solved?

Retrieve. Alice and Bob are supported in their retrieval task in two ways. First, myExperiment.org, BioCatalogue, and the ConceptWiki index their content such that it can be searched through keyword searches. 'Materials and Methods' aggregates stored as myExperiment packs (e.g. packs 82[23] and 58[24]) can thus be found via a familiar search interface. Secondly, when data is exposed as RDF, the Semantic Web query language SPARQL can be used to retrieve precise graphs from the Web of Data. Semantic search can be further facilitated for Alice and Bob if we can hide the SPARQL syntax and the complexity of federated queries through a familiar keyword-based search interface that incorporates auto-completion and browsing of related concepts. As a query language, SPARQL is meant for use by developers so a sufficiently user friendly interface would supply common search patterns, built on a set of SPARQL queries.

[22] http://www.uniprot.org/uniprot/P08592
[23] http://www.myexperiment.org/packs/82
[24] http://www.myexperiment.org/packs/58

Review. The review process is supported, because by using RDF to expose data and to link data it is now possible for Bob to query the complete evidence graph from hypothesis to input to experiment to output to interpretation. In section 3, we demonstrated this principle by retrieving the service that produced one of the proteins in our result set and the creator of that service. Bob can further explore the meaning of Alice's results by exploring any additional links contained in our RDF resources. For instance, we can retrieve extra information from UniProt.

Repeat, reuse, repurpose. Workflows are particularly useful to repeat, reuse, or repurpose a bioinformatics analysis pipeline. A workflow created with a workflow system like Taverna can be reused in new designs. Semantic annotation as facilitated by the AIDA plugin, which gives extra information about the intent of the workflow, which in turn makes it easier to reuse. The particularly tricky bottleneck by changes in services or their underlying data can be partially addressed by a forthcoming feature of BioCatalogue that will indicate when a service has changed its interface or its behavior. It also indicates whether the service is up and running.

Conserve. Embedding data and models in semantic models exposed as RDF/Linked Data provides Alice and Bob with an alternative way to publish and share information. Using identifiers from the Concept Web further lowers the threshold to link information across the web, and to study those links. When myExperiment packs can be accessed via RDF as Research Objects with a consistent interface across the world we have successfully created to new paradigm for scientific publications.

4.1 Research Objects for Publication

The myExperiment 'pack' provides a mechanism for bundling together a collection of resources. The pack is relatively simple in terms of structure, however, essentially providing a "zip file" containing resources, or references to resources. The pack itself can then be annotated with appropriate metadata and shared through myExperiment. As discussed above, the relationships between the resources involved in an experiment (data, methods, results, provenance) is much richer than a simple collection. Packs can thus be seen as a first approximation to a 'Research Object' (RO), a mechanism for publishing reproducible research that is shared on the Web [21]. An RO provides a container for the aggregation of resources, along with information about the relationships between those resources. For Alice, an RO contains all the artefacts that Alice would consider a complete 'experiment', while for her peer Bob it contains everything he needs to reproduce the experiment. As discussed in [21], ROs then provide support for reusability, allowing replay of experiments, repetition of experiments and repurposing of experiments, building on the methods and materials employed. As future work, ROs that have been described with ontological annotations could strengthen the validation and review part of our scenario and provide a self-contained set of procedures and accompanying resources.

Acknowledgements

We thank Katy Wolstencroft and Andrew Gibson for suggestions and critically reading the manuscript, and the teams of the myGrid project, myExperiment,

BioCatalogue, the Concept Web Alliance and the Netherlands BioInformatics Centre (NBIC) for their support.

References

1. Gentleman, R.C., Carey, V.J., Bates, D.M., Bolstad, B., Dettling, M., Dudoit, S., Ellis, B., Gautier, L., Ge, Y., Gentry, J., Hornik, K., Hothorn, T., Huber, W., Iacus, S., Irizarry, R., Leisch, F., Li, C., Maechler, M., Rossini, A.J., Sawitzki, G., Smith, C., Smyth, G., Tierney, L., Yang, J.Y.H., Zhang, J.: Bioconductor: open software development for computational biology and bioinformatics. Genome Biology 5, R80 (2004)
2. Wilkinson, M.D., Links, M.: BioMOBY: an open source biological web services proposal. Briefings in Bioinformatics 3, 331–341 (2002)
3. Goble, C.A., Bhagat, J., Aleksejevs, S., Cruickshank, D., Michaelides, D., Newman, D., Borkum, M., Bechhofer, S., Roos, M., Li, P., De Roure, D.: myExperiment: a repository and social network for the sharing of bioinformatics workflows. Nucleic Acids Research (2010)
4. Bhagat, J., Tanoh, F., Nzuobontane, E., Laurent, T., Orlowski, J., Roos, M., Wolstencroft, K., Aleksejevs, S., Stevens, R., Pettifer, S., Lopez, R., Goble, C.A.: BioCatalogue: a universal catalogue of web services for the life sciences. Nucleic Acids Research (2010)
5. Mons, B., Ashburner, M., Chichester, C., van Mulligen, E., Weeber, M., den Dunnen, J., van Ommen, G.J., Musen, M., Cockerill, M., Hermjakob, H., Mons, A., Packer, A., Pacheco, R., Lewis, S., Berkeley, A., Melton, W., Barris, N., Wales, J., Meijssen, G., Moeller, E., Roes, P.J., Borner, K., Bairoch, A.: Calling on a million minds for community annotation in WikiProteins. Genome biology 9, R89 (2008)
6. Neumann, E., Miller, E., Wilbanks, J.: What the semantic web could do for the life sciences. Drug Discovery Today: BIOSILICO 2, 228–236 (2004)
7. Marshall, M., Post, L., Roos, M., Breit, T.: Using Semantic Web Tools to Integrate Experimental Measurement Data on Our Own Terms. In: On the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops, pp. 688–679 (2006), http://dx.doi.org/10.1007/11915034_92
8. Bizer, C., Heath, T., Berners-Lee, T.: Linked Data. International Journal on Semantic Web and Information Systems 5 (2009)
9. Bizer, C., Heath, T., Idehen, K., Berners-Lee, T.: Linked data on the web (LDOW 2008). In: Proceeding of the 17th international conference on World Wide Web - WWW 2008, Beijing, China, p. 1265 (2008)
10. De Roure, D., Goble, C., Bhagat, J., Cruickshank, D., Goderis, A., Michaelides, D., Newman, D.: myExperiment: Defining the Social Virtual Research Environment (2008)
11. Newman, D., Bechhofer, S., Roure, D.C.D.: MyExperiment: An Ontology for e-Research. In: Proceedings of the Workshop on Semantic Web Applications in Scientific Discourse (SWASD 2009), Washington DC, USA (2009)
12. Missier, P., Sahoo, S., Zhao, J., Goble, C.A., Sheth, A.: Janus: from workflows to semantic provenance and linked open data. In: Proceedings of The Third International Provenance and Annotation Workshop, Troy, NY, U.S.A (2010)
13. Zhao, J., Miles, A., Klyne, G., Shotton, D.: Linked data and provenance in biological data webs. Briefings in Bioinformatics 10, 139–152 (2009)
14. Sahoo, S., Sheth, A.: Provenir ontology: Towards a Framework for eScience Provenance Management. In: Microsoft eScience Workshop, Pittsburgh, PA, USA (2009)

15. Noy, N.F., Shah, N.H., Whetzel, P.L., Dai, B., Dorf, M., Griffith, N., Jonquet, C., Rubin, D.L., Storey, M., Chute, C.G., Musen, M.A.: BioPortal: ontologies and integrated data resources at the click of a mouse. Nucleic Acids Research 37, W170–W173 (2009)
16. Luciano, J.S., Stevens, R.D.: e-Science and biological pathway semantics. BMC Bioinformatics 8(suppl 3), S3 (2007)
17. Mons, B., Velterop, J.: Nano-Publication in the e-science era. In: Proceedings of the Workshop on Semantic Web Applications in Scientific Discourse (SWASD 2009), CEUR-WS, Washington DC, USA, p. 14 (2009)
18. Roos, M., Marshall, M.S., Gibson, A.P., Schuemie, M., Meij, E., Katrenko, S., van Hage, W.R., Krommydas, K., Adriaans, P.W.: Structuring and extracting knowledge for the support of hypothesis generation in molecular biology. BMC bioinformatics 10(suppl. 10), S9 (2009).
19. Clark, T., Kinoshita, J.: Alzforum and SWAN: the present and future of scientific web communities. Briefings in bioinformatics 8, 163–171 (2007)
20. Smith, B., Ceusters, W., Klagges, B., Köhler, J., Kumar, A., Lomax, J., Mungall, C., Neuhaus, F., Rector, A.L., Rosse, C.: Relations in biomedical ontologies. Genome Biology 6, R46 (2005)
21. Bechhofer, S., De Roure, D., Gamble, M., Goble, C., Buchan, I.: Research Objects: Towards Exchange and Reuse of Digital Knowledge. In: The Future of the Web for Collaborative Science (FWCS 2010), Workshop at WWW 2010, Raleigh NC (2010), http://precedings.nature.com/documents/4626/version/1

Towards More Adaptive Voice Applications

Jörg Ott

Aalto University
School of Science and Technology
Department of Communications and Networking
jo@comnet.tkk.fi

Abstract. With the Internet designed to provide best-effort packet transmission, applications are expected to adapt dynamically to the operating conditions observed in the network. For this purpose, congestion control mechanisms have been devised for various transport and (partly) application protocols, and application programs may present, e.g., data rate information to the user. While these mechanisms work well for elastic applications (such as file transfer), the perceived performance of real-time applications may degrade quickly if a minimum required quality of service cannot be achieved. We argue that the current interpretation of adaptation specifically of real-time applications is too narrow and present a framework for expanding the scope of end-to-end adaptation, using the case study of voice communications. Our approach is general in nature, but should especially support communication in mobile environments.

1 Introduction

The Internet Protocol inherently offers a best-effort datagram delivery service, allowing for arbitrary delay, reordering, loss, and duplication of packets. While reordering and duplication occur but are not (yet) commonplace, delays and losses are inherent properties used in the operation of many Internet protocols, as they serve as a measure for congestion. Transport and application protocols (should) monitor these values and are supposed to dynamically adapt their behavior to the changing network conditions.

While transport protocols may be designed to adapt as needed (see section 2), two related assumptions are implied for the applications involved: A1) They are capable of adapting across a wide range of transmission characteristics. In practice, however, the only truly adaptive applications appear to be file transfer, be it as a simple client-server system or a more sophisticated peer-to-peer sharing application, and other ones operating in the background without being time critical. Instead, most applications have a limited operational range of network characteristics acceptable to the user, leading to: A2) The best effort service achieved will be sufficient for the application needs. If it is not, users will notice and react and typically stop using the applications. This applies to (semi-)interactive TCP-based applications (such as web access, HTTP streaming, and SSH) and even more so to UDP-based real-time applications such as VoIP.

We use the term *elastic applications* for those that are capable of adapting as described above (at least within very broad limits), and *inelastic applications* to denote those that are limited to a narrower range of operating conditions: due to their very

T. Margaria and B. Steffen (Eds.): ISoLA 2010, Part I, LNCS 6415, pp. 355–366, 2010.

nature (e.g., real-time monitoring) or because of (present) user expectations (e.g., conversational voice).

In this paper, we broaden the scope of application adaptivity in two ways using voice communication as a case study: a) We capture a broader range across which to adapt, specifically extending to high delays and temporary disconnections so that networking conditions are less critical for a voice conversation. b) To support this broader range, we include more of the application semantics into the considerations of the adaptation process: In our specific case, we embrace asynchronous voice messaging and semi-synchronous (two-way alternate) walkie-talkie-style communications and make these become an integral part of a voice conversation as we will discuss below.

Our goal is to conceptually capture those cases in which the above assumption A2) no longer holds and to devise exemplary mechanisms to fulfill assumption A1) nevertheless. After a short review of related work on adaptive communication (focusing on congestion control) in section 2, we describe mobile communication environments as one case in which assumptions A1) easily fails in section 3. We present the case study for voice communications in section 4, using two examples of our prior work addressing how to maintain A2), from which we then develop a more general adaptation mechanism. We conclude with a brief discussion in section 5.

2 Background and Related Work

At the transport layer, the notion of congestion control has been introduced to TCP [Jac88] (with countless variants developed since) to share network resources fairly (at the flow level), a concept incorporated in recent protocols such as SCTP [Ste07] and DCCP [KHF06a, KHF06b]. Transport protocols also perform timeout adaptation (e.g., TCP's RTO) to cope with varying delays across different networks, which may span more than six orders of magnitude between a local Ethernet link and a loaded GPRS network. Path MTU discovery [MD90] is used to determine an appropriate packet size (e.g., TCP's segment size) in order to avoid fragmentation, although today often 1460 bytes are assumed to be safe.

All these mechanisms are hidden inside the transport layer and thus invisible to the applications (figure 1 (left)). The underlying assumption is that applications using such transport protocols are sufficiently elastic, i.e., can deal well with changing transmission rates and delays. This abstraction was criticized in the past (e.g., [MBL+04]), as TCP would retransmit potentially outdated information and an application is not able to determine the detailed status of the send buffer. The channel concept of SCTP and its unreliable mode, developments such as *Structured Streams* [For07], and more direct resource control [MBL+04] would allow for some more flexibility. Irrespective of these limitations, several applications use TCP to carry real-time data (e.g., skype uses TCP as a fallback) to simplify/enable NAT traversal, with some adaptation realized on top [GKLW02, BBRS08].

Inelastic applications, in contrast, have often avoided TCP and realized the necessary protocol mechanisms on top of UDP, leaving full control to the application as shown in figure 1 (right). This applies to performing semantic fragmentation of application data units and their mapping onto packets [CT90] as is common for real-time

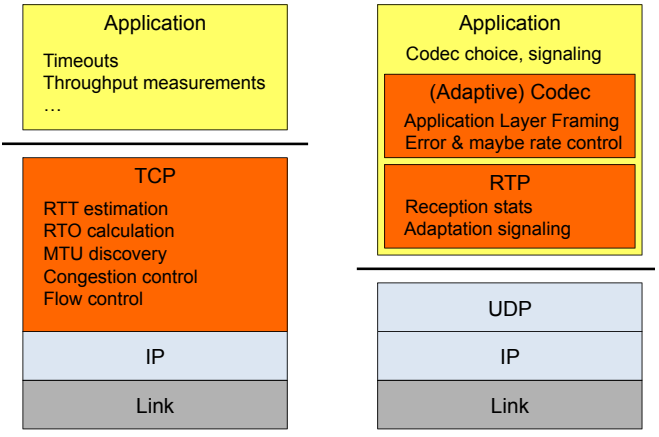


Fig. 1. Extreme cases for adaptivity in today's protocol stacks

media transmissions using RTP [SCFJ03, HP99]. Real-time applications are also supposed to adapt to network congestion and packet losses by means of rate adaptation and error repair and concealment techniques. Rate adaptation can be performed, e.g., by switching codecs [BVG96], changing codec parameters as for multi-rate codecs such as AMR [SWB01], and by adjusting packet sizes. Error control may involve interleaving [PHH98], dynamically applying FEC [PKH+97, Li07, BFPT99] or retransmissions [RLM+06, KH04, ÁHI08]. Several of the above mechanisms may also be combined (e.g., [MPLJ03]).

The necessary feedback loops for observing network conditions to allow a sender to dynamically adapt its transmission behavior may be based upon RTCP reporting [SCFJ03, OWS+06, Gar07]. Such feedback loops may also use information of transport protocols such as DCCP [Per07, BENB07] (or SCTP), in which case the solutions are in-between the two extremes shown in figure 1.

Overall, the key consideration for adaptivity in interactive real-time communications remains the perceivably acceptable delay when trading off delay, loss, and data rate in rate control, interleaving, FEC, and retransmission schemes. According to ITU-T G.114 [Int], the one-way delay for interactive voice communication should not exceed 150 ms, 150–400 ms are potentially tolerable, and delays above 400 ms are not acceptable. This limits the range of network conditions across which these applications can operate *if the notion of interactive voice shall be preserved.*

3 The (Mobile) Internet Today: When Best Effort Is Not Enough

With wireless and mobile Internet becoming increasingly dominant, the connectivity characteristics for mobile users deserve explicit consideration in protocol design [Ott06]. It is well known that wireless and mobile connectivity characteristics may differ significantly from what is observed in the fixed Internet. Specifically, wireless links are susceptible to attenuation, interference, etc. as well as varying load due to shared

channels and may yield highly variable delays (jitter) due to link layer retransmissions and queuing. Worse, the physical layer phenomena and also coverage gaps may lead to temporary loss of connectivity: from fractions of a second to minutes or hours.

In effect, loss, delay, and throughput of wireless links may vary more heavily and more abruptly than in fixed networks, making adaptation more demanding. While congestion in fixed networks often builds up gradually and thus makes rate, loss, and delay estimation somewhat predictable (unless routing changes), sudden interference, handovers to networks with (much) lower performance, or coverage gaps may impact (and possibly stall) communication instantaneously; similarly, conditions may improve suddenly and capacity becoming available again may not be fully exploited.[1] To cope with short disconnections (specifically during handovers), network operators often perform significant queuing for their mobile data networks; however, this leads to delay being accumulated, negatively impacting TCP and interactive application protocols. On the other hand, if no or little queuing takes place, more losses occur. Longer disconnections will lead to packet losses in either case. Overall, the outcome from an application perspective is delay [Ott08].

Interactive voice applications suffer from high delays and jitter as discussed above. Jitter requires playout buffering and thus may incur additional delays (if adapted too conservatively); when delays exceed the predictions for playout buffering so that packets miss their playout deadline, these packets are discarded, increasing loss.

In contrast, VoIP can adapt quite well to the available bit rate. Since a range of codecs exists nowadays (some of them support variable data rates), audio communication can be adapted between some 4 kbit/s (Speex, AMR-NB) and 64 kbit/s (G.711), providing a flexible operating range to cope with congestion. If the short-term average data rate drops below the lower bound, transmitting speech is no longer possible in real-time and additional delay is introduced—or packet losses occur. However, delay and round-trip time (RTT) impact the reactivity of the rate adaptation mechanisms and late adaptation may increase queuing delays and/or cause losses.

Occasional packet losses (due to congestion or bit errors) are tolerable for voice applications and can be handled by receiver-side error concealment techniques. More substantial loss rates or loss bursts lead to unintelligible speech and therefore require applying error control mechanisms, all of which increase the overall delay. As it was found for *skype* users that delay has a less significant impact on perceived speech quality than losses [CHHL06], some flexibility for packet repair exists beyond the guidelines of G.114 [Int], so that loss bursts and short-term outages might be concealed.[2]

Nevertheless, as losses, delays or the frequency of disconnections grow, interactive voice becomes essentially unusable. At some point, rate and error control mechanisms

[1] From an endpoint's perspective it is close to irrelevant whether communication is inhibited by congestion or due to mobility (and it may be impossible to find out); hence, a protocol instance at an endpoint should not (need to) care.

[2] It is worthwhile noting that ITU-T recommendation G.114 is from 1993, a time before the widespread acceptance of mobile telephony (with quite different reliability characteristics) and VoIP. User expectations are probably changing with the use of different communication infrastructures and users may be willing to trade off, e.g., the earlier notion of quality for additional value (mobility, reachability, presence and messaging integration) or lower cost.

may no longer be capable of adapting to the environment and the user has to choose between less interactivity and reduced intelligibility.

This example shows that wireless and mobile communication may yield a best effort service that is insufficient for specific applications if the network characteristics are (temporarily) outside the operational range supported by the respective application (in our case conversational voice). In the above example, the voice communication application is limited in its capability to adapt to a broader range of operational conditions because of the fairly strict interactivity requirements: the specific application semantics define what *interactive voice* means to meet the user expectations.[3] Rather than trying to make all networks match the application needs (e.g., using elaborate QoS mechanisms), we suggest to revisit the application semantics and investigate whether those can be broadened to allow expanding the application's operational range.[4]

For voice communication, this could mean reconsidering the notion of *interactivity*: if we are willing to tolerate higher one-way delays well beyond 150–400 ms, we may be able to employ better error control mechanisms and improve intelligibility of speech. And we may be able to deal with (short) connectivity disruptions by temporarily buffering speech and relaying it when connectivity becomes available again. Both would shift voice communications towards less synchronous interactions if the environmental conditions so require. In the following section, we will discuss two distinct extensions to conversational voice applications that explore the aforementioned ideas. We then develop them further towards a concept of a more adaptive voice application that has a broader notion of interactive voice.

4 Case Study: Adaptive Voice Communication

Users of mobile phones are well aware of varying network conditions and the resulting effects ranging from short-term outages to call disconnections. And users of VoIP over wireless networks (such as WLANs) often experience (short) periods of unintelligible speech due to losses. In either case, the voice applications are not capable of adapting to the network conditions and error handling is up to the user: from "manual synchronization" by repeating sentences to re-dialing a lost call [OX07]. We have investigated two different classes of mobile voice communication applications to deal with insufficient connectivity:

1. **Disconnection-Tolerant SIP**

   We have investigated switching between plain SIP voice calls and voice messaging for IP networks, integrating automatic redial functionality [OX07]. In this approach, two endpoints constantly monitor the networking conditions by observing

[3] For other applications, this holds even if the underlying transport (and session) protocols are capable of tolerating a broader operational range (see, e.g., the countless extensions on TCP performance enhancements and disconnection tolerance as discussed in [OSC+09]) and if the user would be willing to tolerate a lower degree of interactivity, simply because application operations may time out when disconnections last sufficiently long.

[4] Recall that, due to the very nature of wireless communications, QoS cannot always be guaranteed, so that the mechanisms discussed here should also be suitable for well-managed wireless networks.

RTP and RTCP reports from their respective peer to detect gaps in connectivity. These gaps are classified into short (up to several seconds of lost speech, but the call continues), medium (up to one minute, call disconnected), and large (more than one minute, call disconnected and could not be re-established). In addition, each endpoint records the last few seconds of transmitted speech in a local ring buffer for later auto-recovery.

Short outages are recovered by automatically repeating the last (likely incompletely received) talk spurt(s) to the peer (which performs duplicate filtering to avoid too much replay) so that the re-synchronization ("What did you say?") is not entirely up to the users. Medium gaps are repaired by automated re-dialing and answering paired with playback so that the parties can continue talking. Long disconnections are addressed by redirecting the last bits of the conversation to the peer's voice mail so that at least the last statements can be completed and stored for later retrieval.

These mechanisms address those cases in which the interactive conversation is already disturbed by the environment and provide a means to simplify recovery.

2. **DT-Talkie: Asynchronous Voice Messaging**

We have developed an asynchronous voice messaging application running on top of delay-tolerant networks (DTNs) [Fal03] that allows peer-to-peer voice conversations for two or more parties, somewhat similar to Push-to-Talk services (PTT), but without reliance on network or server infrastructure and without the need for a real-time path [ITK+09, Isl09]. Its use of DTN concepts makes *DT-Talkie* applicable to ad-hoc network environments as well as disconnection-prone mobile connectivity.

The *DT-Talkie* captures and stores voice statements locally and groups these statements into voice messages. The messages are sent asynchronously to the respective peer, directly or indirectly via intermediate nodes, with each message being transmitted reliably hop-by-hop using the underlying DTN protocol stack. After (complete) reception, the receiver renders the message from local memory; timing is maintained within but not across statements. Capturing to and rendering from local memory ensure smooth recording and playback, the use of a hop-by-hop-reliable transport ensures that no message parts get lost. Together, this decouples the fidelity of voice communication (perfect intelligibility) from the underlying network conditions, at the expense of increasing delay.

This application goes well beyond disconnection-tolerant SIP in assuming non-connectivity in the first place. Well-connected nodes experience roughly similar quality as Push-to-talk-over-Cellular (PoC) services, whereas poorly connected ones are able to communicate better than before.

We can generalize these ideas further and synthesize both approaches if we assume a more flexible communication substrate allowing for the exchange of small to arbitrarily large packets (or: messages)[5], thus creating a continuum of voice-based interactions as shown in figure 2. At the top of the figure, we indicate today's disjoint voice applications

[5] This functionality may be realized at the network layer, as in some DTNs or in a future Internet [Ott08], or at the transport or application layer as an overlay on top of IP.

from real-time interactive voice on one end of the spectrum to voice mail on the other. These disjoint applications represent different ways of interactions between users with different degrees of interactivity and delay tolerance. They could be integrated as different *modes of operation* into a single encompassing voice application as described below.

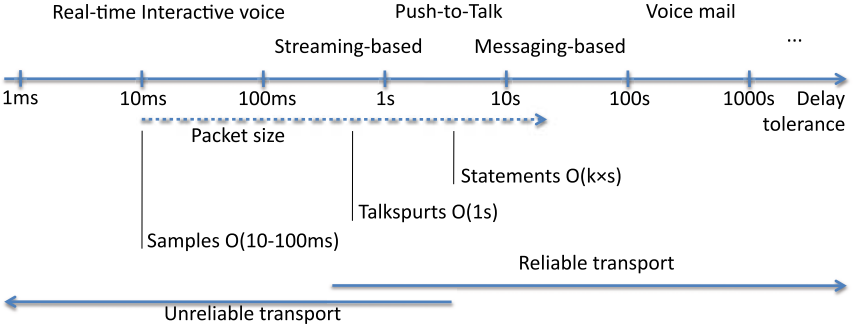


Fig. 2. Adaptivity of voice applications: Different application classes (top) exhibit different degrees of delay tolerance (shown on the axis)—which is related to their packet sizes (shown using the order notation O(...). The larger the packets are, the more important gets reliable delivery.)

As noted above, a key metric for communications is delay, since packet loss may be reduced when allowing for more delay, disconnections can be overcome by waiting sufficiently long, and the data rate of audio codecs appears sufficiently adaptive for most scenarios; if less instant capacity is available, reverting to non-real-time transmission will help, at the expense of increased delay. The mode of operation with the lowest delay are interactive real-time voice conversations, in which we assume minimal mouth-to-ear-delay (typically < 150ms) for acceptable interactivity. Longer delays are tolerable if we move towards less synchronous interactions, as known from one-way alternate communications via walkie-talkies or *Push-to-Talk*; yet some degree of interactivity is preserved. Depending on the network conditions, information exchange may be based upon real-time packet streaming (as in Push-to-talk over Cellular) or reliable exchange of messages (as in our *DT-Talkie*). Finally, voice messaging offers a rather asynchronous style of interaction, more comparable to email.

If we exploit all these different modes of operation and move smoothly between them (subject to the consent of the user), we can make VoIP applications more elastic and expand their operational range. This is conceptually depicted in figure 3. The traditional adaptation for (in this case SIP-based) VoIP systems is shown at the top: The user has some high-level control to specify preferences (usually via quite flexible default settings to ensure interoperability) towards the VoIP application. It couples the media exchange and the signaling functions (call setup and teardown, etc.) and performs the real-time capturing and rendering functions. Based upon the user preferences, the codec and transport layers (their mechanisms are often also referred to as source and channel coding, respectively) perform codec-specific and generic error and rate control. Assuming, e.g., RTCP to monitor RTT, jitter, loss rate, and (implicitly) connectivity, the

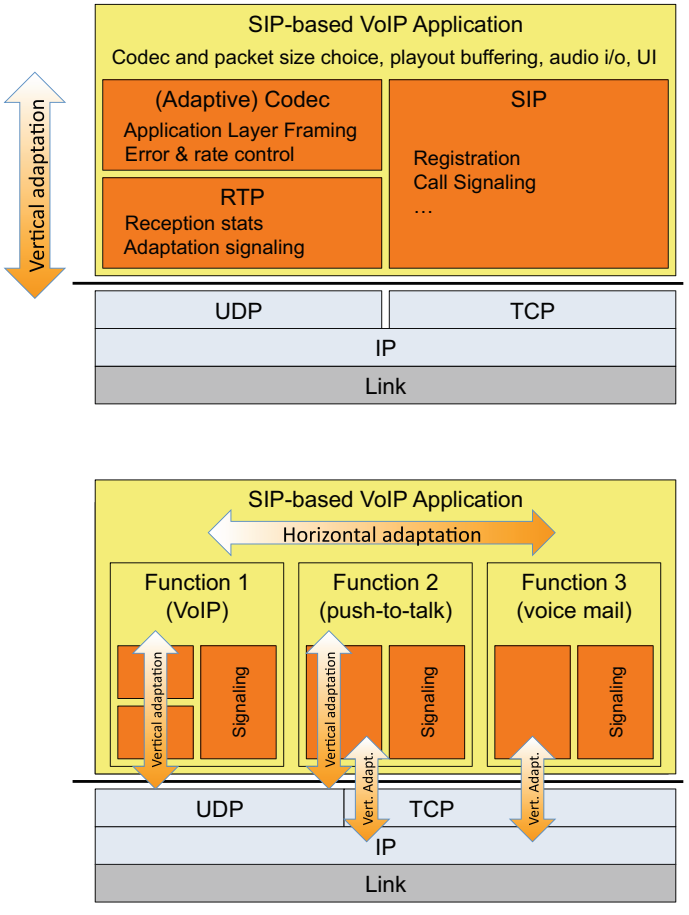


Fig. 3. Vertical only (top) and integrated horizontal (bottom) voice adaptivity

endpoints can adapt their bit rate by choosing different audio codecs and their packet rate (and header overhead) by varying packet sizes.

The entire adaptation is *vertical*, i.e., constrained to the media protocol stack, and after the initial call setup not further intertwined with call signaling. Similar vertical adaptation mechanisms exist for other types of voice interaction as shown in the figure at the bottom: however, no cross-function adaptation is foreseen in today's applications.[6]

The above vertical adaptation mechanisms can be leveraged when broadening the scope of interactive voice communications and smoothly extending it to cover push-to-talk- and voice-message-style communications as well. This is conceptually shown in figure 3 at the bottom, denoted as *horizontal* adaptation. Depending on the observed network conditions, an adaptive application may move between different modes of

[6] Of course, a call setup request may be directed to a voice mailbox; but this happens at the call setup time, before the conversation starts, and is thus not related to adaptation.

operations. Ideally, there are no fixed boundaries as in today's applications, but rather a smooth transition would be foreseen.

As shown in figure 2, starting out with a regular voice call, packets are kept small to achieve a high degree of interactivity, with the above vertical adaptation mechanisms applied. When network conditions worsen, voice application data units (ADUs) can be increased further. With increasing ADU sizes, however, delay increases and communication loses interactivity. While using regular sampling intervals (e.g., 10–100ms) for smaller packets, an application may decide to identify talk-spurts and send (groups of) talk-spurts in ADUs to keep such related information together: words of a sentence and a sequence of sentences of a statement. This ensures that related pieces of each talk-spurt are either delivered in their entirety and can be played back without interruption or are not delivered at all. This can be expanded further to gather complete statements of a user (as with walkie-talkies), e.g., by means of local processing using heuristics, leading further towards asynchronous communication. If connectivity to a peer is lost entirely (for some time), one or more local statements may be aggregated and turned into voice mail. Continuous monitoring of networking conditions should also indicate when the situation is improving, so that the application can move again towards synchronous operation (or resume communication after disconnection).

With increasing ADU size, the impact of a single lost ADU grows: error concealment will work less well for 100ms of missing speech than for only 20ms and if entire statements get lost, the peer may wonder why nobody is responding. Hence, larger ADUs suggest using more reliable transport mechanisms—which are "affordable" since the acceptable delay also increases, thus e.g. allowing for retransmissions.

5 Discussion and Conclusion

The above example presents a conceptual *technically-oriented* view on adaptive applications. We have outlined how broader adaptation could be achieved and sketch how media transmission, monitoring, and to some extent signaling could interact to realize this idea. But the details require further investigation and specific protocol and system designs will need rigorous analytical, simulation, and experimental validation. An interesting technical challenge will be multi-party conversations with the parties connected to each other under different, varying path conditions.

At least equally important, however, is the *non-technical* dimension. While it may be feasible to design such an encompassing adaptive voice application, will the idea of a smooth transition across a very broad operational range be *accepted* by human users? This involves at least two aspects: *usability* and *user expectations*. Concerning the former, a suitable (intuitive, unobtrusive) user interface is required. It has to ensure smooth motion back and forth between more synchronous and more asynchronous styles of interaction, with right level of reactivity. It should offer embedded cues to the user about the present degree of interaction for a given conversation, and it should allow a user to easily control the range of adaptation acceptable for a given conversation. As for the latter, it appears important to steer user expectations: this kind of more adaptive application that is a phone at one instant and turns into a walkie-talkie at the next would probably not appeal someone expecting a phone that always works. Since we have seen

in the aforementioned examples of mobile phones and *skype* that users are able to adapt expectations (including increased delay tolerance) and behavior, we are optimistic that also the idea of broader adaptation mechanisms could be embraced.

Finally, while we offered some intuition on how more comprehensive considerations could help adaptivity of voice communications, similar considerations may be applicable to other real-time and non-real-time applications: For example, media streaming applications already perform pre-buffering to deal with varying networking conditions, a concept that has been explored further to deal with temporary disconnections and could be extended towards broader adaptivity in general, possibly integrating Podcast-style downloads and real-time streaming. Also, web applications could toggle more smoothly between online and offline operation [OK06], provided that the application protocols are adapted accordingly [Ott06].

One interesting follow-up question is whether commonalities can be identified across different applications for common support in future transport protocols. Another one warranting further discussion is what the implications on the present (or a future) networking infrastructure are. We have argued that a future Internet should become inherently more delay-tolerant [Ott08] but, as we discussed above, there is a feature interaction between queuing/buffering ADUs inside the network and the end-to-end control loops of the applications. This may call for limited additional interaction between endpoints and network elements—e.g., providing hints for ADU processing—while maintaining the endpoints independent of the network elements.

Acknowledgments

The research leading to these results has received funding from the European Community's Seventh Framework Programme under grant agreement no 216714.

References

[BBRS08] Brosh, E., Baset, S.A., Rubenstein, D., Schulzrinne, H.: The Delay-Friendliness of TCP. In: Proceedings of ACM SIGMETRICS, pp. 49–60 (2008)

[BENB07] Balan, V., Eggert, L., Niccolini, S., Brunner, M.: An Experimental Evaluation of Voice Quality over the Datagram Congestion Control Protocol. In: Proceedings of IEEE INFOCOM (2007)

[BFPT99] Bolot, J.-C., Fosse-Parisis, S., Towsley, D.: Adaptive FEC-based error control for Internet telephony. In: Proceedings of IEEE INFOCOM, vol. 3, pp. 1453–1460 (1999)

[BVG96] Bolot, J.-C., Vega-García, A.: Control Mechanisms for Packet Audio in the Internet. In: Proceedings of IEEE INFOCOM, vol. 1, pp. 232–239 (1996)

[CHHL06] Chen, K.-T., Huang, C.-Y., Huang, P., Lei, C.-L.: Quantifying skype user satisfaction. In: Proceedings of ACM SIGCOMM, pp. 399–410 (2006)

[CT90] Clark, D.D., Tennenhouse, D.L.: Architectural Considerations for a new Generation of Protocols. In: Proceedings of ACM SIGCOMM, pp. 200–208 (1990)

[Fal03] Fall, K.: A Delay-Tolerant Network Architecture for Challenged Internets. In: Proceedings of ACM SIGCOMM, pp. 27–34 (2003)

[For07] Ford, B.: Structured Streams: a New Transport Abstraction. In: Proceedings of ACM SIGCOMM, pp. 361–372 (2007)

[Gar07] Garai, L.: RTP with TCP Friendly Rate Control. Internet Draft draft-ietf-avt-tfrc-profile-09.txt (July 2007) (work in progress)

[GKLW02] Goel, A., Krasic, C., Li, K., Walpole, J.: Supporting Low Latency TCP-Based Media Streams. In: Proceedings of IEEE IWQoS, pp. 193–203 (2002)

[HP99] Handley, M., Perkins, C.: Guidelines for Writers of RTP Payload Format Specifications, RFC 2736 (December 1999)

[Int] International Telecommunication Union, Telecommunication Sector (ITU-T). Transmission Systems and Media, General Recommendation on the Transmission Quality for an Entire International Telephone Connection; One-Way Transmission Time. Recommendation G.114 (March 1993)

[Isl09] Tarikul Islam, M.: Voice Communications in Mobile Delay-tolerant Networks. Master's thesis, Helsinki University of Technology (TKK), Finland (2009)

[ITK+09] Tarikul Islam, M., Turkulainen, A., Kärkkäinen, T., Pitkänen, M., Ott, J.: Practical Voice Communications in Challenged Networks. In: Proceedings of the ExtremeCom workshop (2009)

[Jac88] Jacobson, V.: Congestion Avoidance and Control. In: Proceedings of ACM SIGCOMM, pp. 314–329 (1988)

[KH04] Kropfberger, M., Hellwagner, H.: Evaluation of RTP Immediate Feedback and Retransmission Extensions. In: Proceedings of the IEEE International Conference on Multimedia and Expo (ICME), vol. 3, pp. 1751–1754 (2004)

[KHF06a] Kohler, E., Handley, M., Floyd, S.: Datagram Congestion Control Protocol (DCCP). RFC 4340 (March 2006)

[KHF06b] Kohler, E., Handley, M., Floyd, S.: Designing DCCP: congestion control without reliability. In: Proceedings of ACM SIGCOMM, pp. 27–38 (2006)

[Li07] Li, A.: RTP Payload Format for Generic Forward Error Correction. RFC 5109 (December 2007)

[MBL+04] Mogul, J., Brakmo, L., Lowell, D.E., Subhraveti, D., Moore, J.: Unveiling the Transport. ACM SIGCOMM Computer Communication Review 34(1), 99–105 (2004)

[MD90] Mogul, J., Deering, S.: Path MTU discovery, RFC 1191 (November 1990)

[MPLJ03] Matta, J., Pépin, C., Lashkari, K., Jain, R.: A Source and Channel Rate Adaptation Algorithm for AMR in VoIP Using the Emodel. In: Proceedings of ACM NOSSDAV, pp. 92–99 (2003)

[OK06] Ott, J., Kutscher, D.: Bundling the Web: HTTP over DTN. In: Proceedings of WNEPT (2006)

[OSC+09] Ott, J., Seifert, N., Carroll, C., Wallbridge, N., Bergmann, O., Kutscher, D.: The CHIANTI Architecture for Robust Mobile Internet Access. In: Proceedings of the 10th IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM), Industry Track (2009)

[Ott06] Ott, J.: Application Protocol Design Considerations for a Mobile Internet. In: Proceedings of the 1st ACM MobiArch Workshop, pp. 75–80 (2006)

[Ott08] Ott, J.: Delay Tolerance and the Future Internet. In: Proceedings of the 11th International Symposium on Wireless Personal Multimedia Communications (2008)

[OWS+06] Ott, J., Wenger, S., Sato, N., Burmeister, C., Rey, J.: Extended RTP Profile for Real-time Transport Control Protocol (RTCP)-based Feedback (RTP/AVPF). RFC 4585 (July 2006)

[OX07] Ott, J., Xiaojun, L.: Disconnection Tolerance for SIP-based Real-time Media Sessions. In: Proceedings of the International Conference on Mobile Ubiquitous Multimedia (MUM), pp. 14–23 (2007)

[Per07] Perkins, C.: RTP and the Datagram Congestion Control Protocol. Internet Draft draft-ietf-dccp-rtp-07.txt (June 2007) (work in progress)

[PHH98] Perkins, C., Hodson, O., Hardman, V.: A survey of packet loss recovery techniques for streaming audio. IEEE Network, 40–48 (September/October 1998)

[PKH+97] Perkins, C., Kouvelas, I., Hodson, O., Hardman, V., Handley, M., Bolot, J.C., Vega-Garcia, A., Fosse-Parisis, S.: RTP Payload for Redundant Audio Data. RFC 2198 (September 1997)

[RLM+06] Rey, J., Leon, D., Miyazaki, A., Varsa, V., Hakenberg, R.: RTP Retransmission Payload Format. RFC 4588 (July 2006)

[SCFJ03] Schulzrine, H., Casner, S., Frederick, R., Jacobson, V.: RTP: A Transport Protocol for Real-Time Applications, RFC 3550 (July 2003)

[Ste07] Stewart, R.: Stream Control Transmission Protocol. RFC 4960 (September 2007)

[SWB01] Seo, J.W., Woo, S.J., Bae, K.S.: A study on the application of an AMR speech codec to VoIP. In: Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), vol. 3, pp. 1373–1376 (2001)

[ÁHI08] Huszák, Á., Imre, S.: DCCP-based Multiple Retransmission Technique for Multimedia Streaming. In: Proceedings of the 6th ACM International Conference on Advances in Mobile Computing and Multimedia (MoMM), pp. 21–28 (2008)

Telco Service Delivery Platforms in the Last Decade - A R&D Perspective

Sandford Bessler

FTW Telecommunications Research Centre Vienna
bessler@ftw.at

Abstract. This overview discusses the technological and architectural evolution of the telco service delivery platforms in the last decade. We argue that not only the telco business model but also the system environment contributed to the decrease in revenues from added value services and continues to be threatened by the success of the OEMs and their AppStore model. Finally, we review a number of areas in which network operators continue to play a crucial role in the service value chain.

Keywords: service delivery platforms, Telco, IMS architecture, enablers, OSA Parlay, AppStore, business model, pricing, service creation.

1 Introduction

Twelve years ago Sun Microsystems, in cooperation with innovative network operators such as BT, announced to open the interfaces of the intelligent network (IN) for the development of 3rd party services. The initiative, called JAIN, was the first in a series of technological attempts to generate added value from the telco capabilities, user data and network information. The prospective advantage to interwork with different existing variants of the largest (telephony) network at the time via a standardized API and leverage the value of applications written once and running everywhere lead to a joint standardization group of ETSI, 3GPP and Parlay that developed the so called Parlay/OSA model. Based on capabilities such as location, messaging, presence and call control, they developed CORBA interfaces that were powerful but difficult to use by the mass web application programmers. Later, about a dozen interfaces called Parlay-X were defined using the web service technology (WSDL bindings). Parlay "light" had a higher abstraction level, had fewer callbacks than the CORBA interfaces, fewer complex types and therefore it was easier to use.

In 2004, the mobile telco standardisation body 3GPP developed the IP Multimedia Subsystem (IMS), based on the service initiation protocol (SIP), and tried to integrate the Parlay service capabilities with the new defined SIP application servers in a new IMS service architecture (see Figure 1).

At this point, the design, pushed by the huge 3GPP group, began to show weaknesses: whereas the Parlay capabilities were functionally isolated and divided into call control, presence, messaging, etc.), the SIP signalling mixed all

T. Margaria and B. Steffen (Eds.): ISoLA 2010, Part I, LNCS 6415, pp. 367–374, 2010.

these functionalities in one protocol: call control using the INVITE message, messaging using MESSAGE and presence using SUBSCRIBE (NOTIFY messages (see Figure 1).

Concerning the interface to 3rd party applications, the openness achieved in JAIN and Parlay became in IMS a walled garden again, as the network operators were reluctant to disclose to every small service provider both a SIP (ISC) interface and the authentication DIAMETER interface to their main subscriber database.

The IMS (NGN) design followed the mobile intelligent network (CAMEL) principle according to which a mobile user can be either in a home or visited network. This very successful design for simple services such as telephony created many problems for complex IMS services: for example, a location based service that is discovered in the visited network (the well known nearest pharmacy example) is in IMS still controlled by the home network of the user - today, in the era of global internet services, an unnecessary complexity.

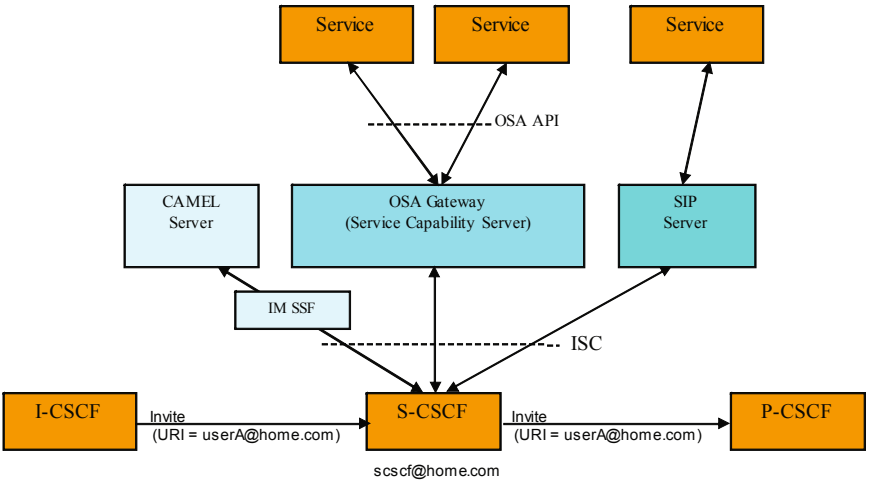


Fig. 1. IMS Service architecture. The call state control functions (CSCF) are SIP routing components.

Unfortunately, the initial use of IMS service delivery platforms, the so called enablers and their open interfaces to the applications failed to bring the expected revenues. According to Manuel Vexler from Huawei [3], in the LTE future network, IMS would play an important role in routing voice calls between the IP and circuit-switched networks.

A thorough analysis of the reasons for this situation goes beyond the scope of this paper; the remarks below reflect our experience from a R&D perspective:

– Network operators had high expectations from collecting, processing presence information from several sources (user, network, etc..) and selling it as a value added service. Presence is a form of context and can be used to redirect

a call, adapt a service or just inform another user. On the one hand the rich presence IETF standard is complex, allowing many presence states (location type, activity type, even mood of a user), on the other hand it is not extensible to new forms of presence (see [1] for an idea how to avoid that). Within the IMS infrastructure, presence information has to be conveyed via the SIP protocol. As a consequence, many web companies, providers of instant messaging and social network applications bypassed the IETF and SIP protocols and use today XMPP or proprietary dialogues between the terminals and their web 2.0 platforms.

– Much research effort has been put into the convergence of telecommunication related services, that are triggered for example by SIP messages, and web service platforms . The goal is, to achieve flexibility in the service creation process by using workflow engines and scripting, instead of a static, compiled code (see [2] for more details). Although the workflow logic decides what shall happen with a SIP call in real-time, such hybrid applications represent a small market niche, relevant for professional voice applications.
– A Web 2.0 main innovation is to support user generated content. Extrapolating this approach to services, different researchers proposed that users should create their own services on the fly. An innovative EU FP7 project called mCiudad started in 2007 with the goal to demonstrate that users (prosumers) can create simple peer to peer services in a matter of minutes, in and with their mobile terminals. Figure 2 shows the building blocks of the mCiudad system according to [5]: each mobile terminal can access a central

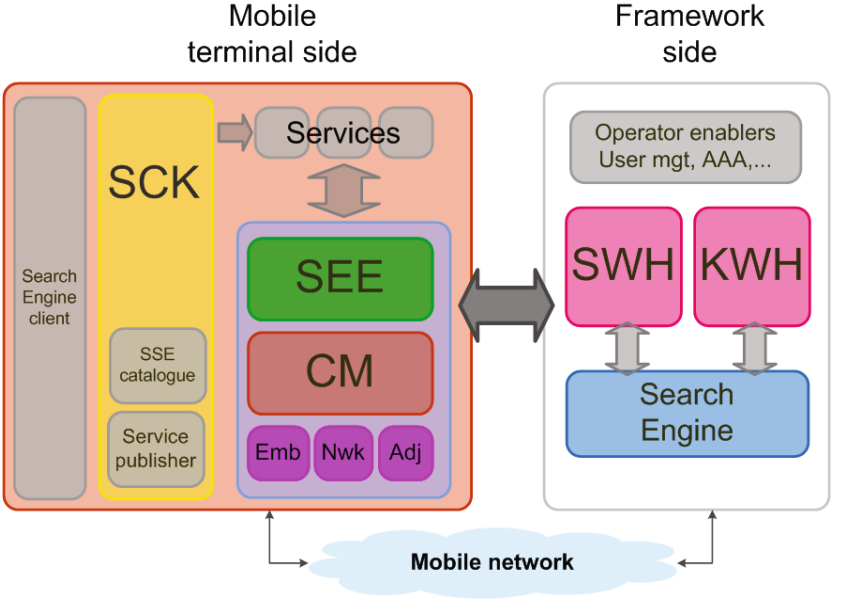


Fig. 2. mCiudad project: service

warehouse of service components (SWH), assembly and personalize them locally using a service creation toolkit (SCK). After the creation the service runs in the service execution environment using a number of local capability functions. The service creation on the own mobile phone is quite challenging in several ways: in defining the communication pattern between the peers, in selecting and matching the right components, in defining the life-cycle of an assembled service and, finally in providing usability for non-experienced users. Therefore, at the end of the project it will be worth to revisit the fundamental question: shall services be created and tested by normal users or by software developers ? According to the adopters of the appStore business model (see next section) that emerged during the runtime of the mCiudad project, a user shall be able to find an app for almost any function she needs. The open questions which are currently investigated in mCiudad relate to intelligent search methods for the desired functionality among a huge base of applications by using for example semantic techniques.

Summarizing, the state of the art delivery platforms based on IMS/NGN failed to create the expected revenues. There were not only flaws in the technical integration of third parties, the time to market and innovation remained below the competing web companies.

2 The New Service Delivery Platforms

In 2008 Apple has introduced the app-store business model for its iPod and iPhone products. The apps are downloadable programs that the users buy for a low price of $0.99 in average. The apps are divided into categories such as: education, economy, entertainment, games, life-style, music, navigation, news, social networks and many more. Note the contrast between the diversity of these application categories and the limited scope of telco oriented applications that have been developed in research and industrial projects in the last decade. Call control, presence, are in the best case integrated in Skype or other VoIP apps, location and navigation use the GPS local API, messaging and email are ubiquitous.

In order to create the apps, programmers mostly deal with the local objects in a SDK. The protocol with the backend servers is often based on HTTP, web service APIs, or proprietary media streaming protocols.

The business model has been adopted by 50 small and large OEMs such as Google, Nokia, RIM, and major network operators. Crucial for the success of the model have been the quality control before the admission into the app store and the agreement to share of up to 70% of the revenues with the app programmers. Vodafone for instance plans to increase its digital value chain by offering the developers access to user subscriber location information, to enable geographically targeted applications that reach the entire Vodafone customer base (30mio. iPhone users), to open up a "billing API" - effectively giving application developers the ability to charge customers directly on their Vodafone bill or prepaid

account. The challenge for Vodafone is that their app store will need to support multiple different devices and technologies.

Besides the mobile phone business, apps started to infiltrate other application domains: cars, electronic devices, scales, home appliances.

According to [6] the market of mobile apps will grow from 6,5 bio. USD in 2010 to 18 bio USD in 2012. Despite the expected growth, there is a lot of discussion and debate in the industry on the topic of Mobile Apps versus Mobile Web in the industry. The critical factors to have an impact of the future of apps are:

1. Penetration of HTML5+ browsers on mobile
2. Difference between the native OS support and browser access to the same APIs
3. Implementation differences between various browsers

The interesting question is however, whether the Telcos should imitate Apple and Co. and build own AppStores or concentrate on their core business, namely to provide high quality delivery for the exploding amount of (mobile) data? In the next section we present a few directions from past and current research.

3 ...and the Network Operators?

The Telco 2.0 initiative of STL Partners [7] is trying to draw new strategies for Telcos by proposing a new, two-sided business model, meaning that the revenues shall originate both from the customers and from the content providers. Since the former "upstream" suppliers, such as content and application providers bypass the Telcos and sell directly to the customers (over the top), the network operators must find ways to charge the content/Internet companies for accessing 'their pipes'. Seven areas for achieving added value are proposed by Telco 2.0: 1) identity, authentication and security, 2) advertising, 3) e-commerce sales, 4) order fulfilment offline and 5) online (e-content), 6) billing and 7) customer care.

Although it will be very hard to compete in all these areas, we think that the 2-sided business model will allow sustainable investment in the broadband infrastructure and will provide quality, mobility, security, enhanced privacy.

The following service examples focus on efficient management of internal broadband resources:

3.1 Application Layer Multicast of Video on Demand Streams

Despite the continuous expansion of backbone and access broadband infrastructure, an internet wide deployment of video on demand service with full size movies poses a formidable challenge for network operators. Meanwhile, most cable and network providers have adopted a closed model that serves the own user population, is based on a relatively small number of local and popular content titles, but enables the full control of the content distribution process. On an internet scale, among the P2P approaches for content distribution networks (CDN), the most successful one uses the BitTorrent technology. Instead of downloading

the same content from the server for each user again and again, it is theoretically possible to do it once, the further distribution being performed autonomously by the participating peers. From a network operator perspective however, the performance level of such a system cannot be managed.

In this scenario we address a video content distribution system for residential customers similarly to the one used with BitTorrent: large content files (movies) are downloaded directly to the user disks, whereas the playback occurs locally. In order to scale up to tens of thousands of movie titles arriving from video servers world wide, we propose to use multicast. Since IP multicast deployment is difficult, the solution approach is based on application layer multicast, a technique operating in an overlay (peer to peer) layer in the network operator domain. Optimization heuristics periodically schedule the multicast trees and to pack them in the available bandwidth, improving the efficiency and scalability of the system.

3.2 A Location Service with Tunable Privacy

Location based services for mobile users were foreseen to have a bright future, however at least in Europe their spread would raise privacy concerns. The application presented in [4] is a simple friend-locator using GPS localization capability but allowing enhanced privacy protection. The principle is quite simple:

Each mobile user can define location "zones", such as home, work, fitness club, school, etc. Each zone is a circle with a center of known coordinates and a radius. Using the contact lists, the user can configure the set of zones each of her contacts may "see".

If another user wants to see my location, she has to subscribe to this location, if allowed by the configuration step above (see Figure 3). The user will then be notified if I enter or leave that zone. Between the zones I cannot be tracked.

The proposal has several advantages concerning privacy: only the location "names" are transmitted but not the coordinates, the zones can be made fuzzy to include for example the whole city, which can still be useful for colleagues to see if you are abroad, and the disclosure of location is completely under the control of the target (called also presentity).

The clue is the realization: it needs to run a presence server on the mobile phone and an efficient notification mechanism. Many email and web notification (push) protocols exist, the majority are realized by keeping TCP connections

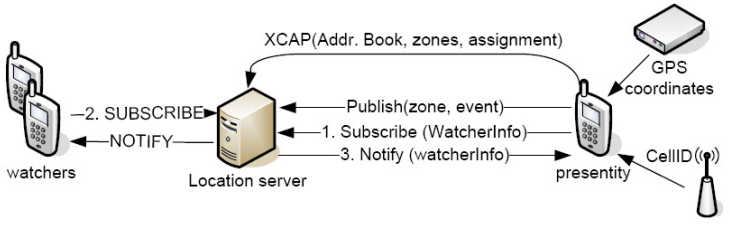


Fig. 3. Privacy enhanced location service with network operator mediation server

open, but this solution is not always reliable. Another way to wake up a phone with an asynchronous notification is to use a SMS/MMS or a SIP message. In the assumption of SIP availability in the smart phones, the network operator could provide the SIP based push service and aggregation functionality, as in the figure below; otherwise, location updates over the air from tens of contacts of a user would increase traffic and drain the battery.

3.3 Quality of Service and Pricing

The question of the revenue of the network operators from content and services is closely related to the network neutrality discussion: mobile data usage is growing rapidly and may cause congestion in the hotspots. The network operators invest large sums in the infrastructure; therefore they should have a share in the revenues.For this to happen, the application high data demands require new pricing models.

Considering the data stream consumed by an end-user, the current situation is that the three variables: pricing, value and usage are independent, or in best case the relation between them is a very simple one: flat price or usage threshold pricing. A better metric for measuring the congestion created by an end-point data stream along the path is needed. One interesting approach [8] is based on the explicit congestion notification (ECN) mechanisms, that works by marking more packets as a queue builds. The share of congestion each end user contributes (congestion volume allowance) can be then priced according to the following use policy, for example 1GB/month (marked packets) for 15 Euro. Bob Briscoe from BT [8] argues that such a system is fair, neutral (no permission needed for QoS) and application agnostic.

In addition, network providers can offer improved connectivity, mobile coverage, better throughput as premium service. They can improve the security and since they are mostly trusted parties, they can offer enhanced privacy on the network level.

According to a study by Freedom Dynamics about QoS/QoE in mobile communications, the first ranked expectation of the 555 respondents was a good internet connection. This shows the crucial role of the network operators, even those who become a (happy) bit pipe: customers pay for QoE and leave for the lack of QoS!

In accordance to the future internet discussion, the network operators have additional options especially in the case of challenged networks where high mobility leads to disruptions in the connectivity. Special enablers help to store the data in the network and to deal with disruptions (disruption tolerance).

4 Concluding Remarks

The impairments in the architecture of the telco service platforms are probably one cause of revenue decrease experience by network operators, the main cause

is however the lack of business models matching the tremendous growth of internet use and the appearance of a service ecosystem that bypasses the network operators services.

Network operators have lost the first round in the battle for revenues from applications but have to maintain their crucial role in the value chain by insuring: quality of the content delivery, optimal resource usage, security, trust, and last but not least, appropriate pricing for network usage.

Research institutes (such as FTW) accompanied the technology and standardization efforts of the telecommunications industry along the way and investigated the internet and telco services, the use of enablers and service creation platforms.

Acknowledgements

This work has been funded by the Austrian Government and the city of Vienna in the COMET program. Many thanks to Manuel Vexler, (Huawei and IMS Forum) for the useful comments.

References

1. Bessler, S., Zeiss, J.: Using Semantic Policies to Reason over User Availability. In: The Second International Workshop on Personalized Networks, Pernets 2007, Philadelphia (2007)
2. Bessler, S., Zeiss, J., Gabner, R., Gross, J.: An Orchestrated Execution Environment for Hybrid services. In: Proceedings Kommunication in Verteilten Systemen (KiVS) 2007, Bern (February 2007)
3. Manuel Vexler, Voice over LTE: Chances and Challenges, presentation at the IMS World Forum (2010)
4. Bessler, S.: A System for Locating Mobile Terminals with Tunable Privacy. Journal of Theoretical and Applied Electronic Commerce Research 2(2), 82–91 (2007) ISSN 0718-1876 Electronic Version
5. Urdiales, D., de las Heras, R., Davies, M., Zhdanova, A.V., Immonen, M., Heinilä, J., Narganes, M., Christophe, B., Maknavcius, L., Barnaghi, P.: m:Ciudad - Unleashing mobile user-provided services. In: WWW 2009, event, Madrid, Spain (April 20-24 (2009)
6. Chetan Sharma, Sizing Up the Global Mobile Apps Market (2010)
7. Telco 2.0 Manifesto, http://www.telco2.net/manifesto/
8. Briscoe, B.: Resolving Internet capacity sharing and neutrality battles, Presentation at TK Forum (2010), http://www.ftw.at/presse-events/telekommunikationsforum/talk_briscoe

Ontology-Driven Pervasive Service Composition for Everyday Life

Jiehan Zhou, Ekaterina Gilman, Jukka Riekki, Mika Rautiainen, and Mika Ylianttila

Computer Science and Engineering Laboratory
Dept. of Electrical Engineering, University of Oulu, PL 4500, FI-90014,
Oulu, Finland
firstname.secondname@ee.oulu.fi

Abstract. Incorporating service composition and pervasive computing into managing user's everyday activities gives rise to the paradigm of Pervasive Service Composition for everyday life. This paper presents a novel generic model for services supporting everyday activities. The resulting service composition consists of local services within service peers and services are executed as specified in peer coordination and service collaboration. We suggest a task-based, pervasive, semantic, and P2P-based approach for service composition for everyday life. We first address these fundamental characteristics. We give terminologies related to service composition, pervasive computing, ontology, and Pervasive Service Composition. Secondly, we analyze requirements for describing everyday activities. To meet the requirements we design an initial ontology model for capturing user's everyday activity and accommodating peer coordination and service collaboration in Pervasive Service Composition. Finally, we classify existing approaches to Web service composition.

Keywords: pervasive computing, service computing, semantic Web services, ontology.

1 Introduction

To reach full potential service composition applications in public domains are expected to have the ability to perform peer-to-peer coordination between the participating service peers. This vision requires service composition to accommodate decentralized service provision and context-aware computing. To meet these requirements, we generalize the concept of Ontology-Driven Pervasive Service Composition (ODPSC), which envisions an ontology-centric solution to flexibly facilitate everyday activity management by forming service peers, consuming individual Web services and integrating Web services within or across service peers in a pervasive computing environment. Ontology is quite often applied in capturing, managing and developing business domain knowledge. Ontology-driven Pervasive Service Composition provides a semantic computing environment for mobile user to manage their complex everyday activities with the emphasis of pervasive computing, task-based computing, service composition, and semantic service composition.

T. Margaria and B. Steffen (Eds.): ISoLA 2010, Part I, LNCS 6415, pp. 375–389, 2010.

Many research and standardization efforts have been made on service computing and service composition to support commercial activities in business domains, e.g., Business Process Execution Language (BPEL[1]) and Choreography Description Language (CDL[2]), WSDL-S[3], IRS-III[4], and SWSF[5] for electronic commercial application integrations. Few studies on Web services are reported for supporting everyday activities in context of pervasive computing. Meanwhile, many literature reviews and comments show us that those studies on service composition are mixed up and confused with orchestration and choreography definitions.

On the other hand, research efforts have been made on pervasive computing for supporting everyday activities. The notion of pervasive computing introduces a vision of user and environments that provide information and services when and where desired [6]. A variety of terms are in use to describe this paradigm, such as ubiquitous computing [7], ambient intelligence [8, 9], etc. However, few studies are reported with incorporating Web services and service composition into pervasive computing.

To integrate service composition and pervasive computing, and distinguish service composition, this paper addresses the fundamental issues in Ontology-Driven Pervasive Service Composition in everyday activity. The remainder of the paper is organized as follows. Section 2 defines terminologies used in this paper in context of Ontology-Driven Pervasive Service Composition. Section 3 analyzes technical requirements for ODPSC. Section 4 proposes an ontology model for Pervasive Service Composition. Section 5 revisits existing standardization activities at syntactic and semantic levels. Section 6 draws conclusions.

2 Fundamental Issues and Terminologies

This section addresses fundamental issues and defines basic concepts related to ontology-driven Pervasive Service Composition. Recent efforts in research and standardization have somewhat blurred the terms related to service composition.

2.1 Fundamental Issues

Ontology-Driven Pervasive Service Composition (ODPSC) is regarded as an advanced means of managing complex everyday activities in a pervasive computing environment. ODPSC can be classified as pervasive, task-based, semantic, and peer-to-peer (P2P)-based computing. These characteristics are discussed next.

ODPSC is utilized to offer pervasive services for users: right services at the right time. The nature of pervasive computing requires ODPSC not only to discover and choose services before composing, but also to discover and regroup service peers dynamically when the situation changes. On the other hand, ODPSC is expected to support user's everyday activities by maintaining Web services and service compositions in a pervasive computing environment. This requires describing the tasks users want to accomplish and composing services for performing these tasks, that is, task-based computing is required. Some services involve only one service party while others involve multiple service parties, namely peers.

ODPSC requires semantics. Service composition involves information exchange – and not only exchanging technical information describing service input/output and

interactions, but also managerial information for service peers and roles. This requires a shared understanding and interpretation of information, including semantic service discovery and composition, and service peer discovery and coordination. Moreover, the operation environment of ODPSC calls for P2P solutions. In traditional business integrations, centralized and less peer-oriented application integration is common in service composition. In contrary, computing appliances assisting everyday activities in pervasive computing environments are loosely coupled and distributed in the environment. Hence collaboration and coordination in everyday activity management requires a P2P-oriented integration, in which the same functional services can be provided by different peers and a peer can provide many different services. The P2P-oriented nature determines the roles of requestors, providers, and registry center in a conventional centralized SOA architecture as service peers. This promotes collaborative and coordinated computing and communication with its advantages of scalability, fault-tolerance, dynamic networking, and collaboration supporting [10-12].

Fig. 1 illustrates how user's everyday activities are coordinated and managed smoothly by applying the ODPSC paradigm. ODPSC facilitates complex activity management in a multi-peer coordinated, intelligent, and cost-efficient way. It requires development of technologies for addressing the above ODPSC characteristics of task-based, pervasive, semantic, and P2P-based computing.

Due to mixed use of concepts and notations in the field of service composition, next section aims to summarize existing term definitions and clarify basic concepts used in context of Ontology-Driven Pervasive Service Composition.

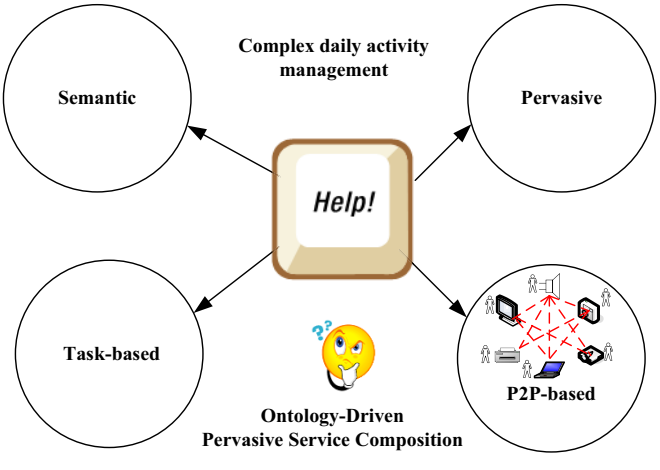


Fig. 1. The Ontology-Driven Pervasive Service Composition paradigm

2.2 Terms in Business Integration

This section lists terms used in the context of business integration. There is a number of terms to describe how components can be integrated together to build complex business processes in industries. Many of the terms in Table 1 have been used in business integration prior to Web services.

With the introduction of Web services, terms such as Web service architecture, Abstract and Executable Process, Web composition, Web service Orchestration and Choreography, etc. have been used to describe business integration (Table 2). Web Service is an autonomous, standards-based component whose public interfaces are defined and described using XML (W3C) and that supports interoperable machine-to-machine interaction over a network using mainly Web-based standards [13][14]. We use the terms "Web service" and "service" interchangeably in this paper. Semantic Web [13] provides the environment and enabling infrastructure for Semantic Web Services that are formally described and semantically enriched Web services.

Web Service architecture stack consists of a series of Web Services specifications, targeting for integrating interacting applications such as primitive layer for basic level applications and contemporary layer for advanced level applications. The primitive layer consists of SOAP, WSDL, and UDDI. SOAP [15] defines the basic messaging protocol independent of message exchanging environments. Web Services Description Language (WSDL) [16] provides a model for describing Web services such as Service, Port, Binding, PortType, Operation and Message. UDDI registry [17] allows publishing the availability of a Web Service and its discovery from service requesters using sophisticated searching mechanisms.

Table 1. Terms used in business integration

Business activity	The activity of providing goods and services involving financial, commercial and industrial aspects [39].
Workflow	Means to handle the routing of work between various resources in an IT organization; the flow or progress of work done by a company, industry, department, or person.
Business process management system	Used to enable a business to build a top-down process design model, consisting of various integration activities; systems typically cover the full lifecycle of a business process, including the modeling, executing, monitoring, management, and optimization tasks.
Contract	An agreement between two or more parties, especially one that is written and enforceable by law [38].
Global contract	Contains a definition of the common ordering conditions and constraints under which messages are exchanged from a global viewpoint, the common and complementary observable behavior of all the participants involved.
Behavior	The set of the actions or reactions of a person or animal in response to external or internal stimuli.
Observable behavior	Visible and public behavior to involving parties.
Non-observable behavior	Invisible and private to involving parties.

The contemporary layer consists of Business Process Languages layer and Choreography layer. Business Process Languages layer describes the execution logic of Web Services based applications by defining their control flows. Choreography layer describes collaborations of participants by defining their common and complementary observable behavior, where information exchanges occur, when the jointly agreed

ordering rules are satisfied from a global viewpoint. The contemporary layer is being extended for accommodating other system characteristics such as security (security layer), reliability (Reliable Messaging layer) and so on. Other terms are as follows:

Table 2. Web Service related terms

Workflow	Defines the way work can be done from the beginning to the end, including sequences of tasks together with rules that must be followed.
Process	A set of interrelated activities that constitute a complete task.
Activity	A task forming one logical step of a process.
Internal or private business process	Business behavior acted by internal decision making, not revealed to business partners. For example, procurement logistic made by the seller. Such business process is also called as non-observable behavior.
External or public business process	Business behaviors agreed by involved business parties. For example, returning policy of goods made between customers and sellers. Such business process is also called as observable behavior.
Abstract processes	Serves a descriptive role to describe observable message exchange behavior of each of the parties involved, without revealing their internal implementation. Executable processes are fully specified and thus can be executed. [18]
Web service composition	Provides the means for business integration by weaving Web services.
Web Services composition language	A language for the description of Web Services compositions.

There are two types of Web Services composition languages. One is programming in the large, which generally refers to the description of the high-level state transition interactions of a process such as choreography language. Another type is programming in the small, which deals with short-lived programmatic behavior involving local logic processing such as orchestration language.

Orchestration language specifies an executable process that involves message exchanges with other systems, such that the message exchange sequences are controlled by the orchestration designer. An orchestration language describes how Web services interact with each other at the message level. For orchestration, the process is always controlled from the perspective of one of the business parties.

Choreography language specifies a protocol for peer-to-peer interactions, tracks the sequence of messages that may involve multiple parties and multiple sources, including customers, suppliers, and partners. A choreography language describes how Web services interact with each other at the peer level. A choreography description is the multi-participant contract that describes a composition from a global perspective; their decision making and data management are observable for all.

2.3 Terms Related to Ontology-Driven Pervasive Service Composition

Ontology-Driven Pervasive Service Composition provides a pervasive computing environment for facilitating everyday activities. This paradigm is characterized with task-based, pervasive, semantic and P2P-based computing.

The quality of human everyday life (e.g. productivity and creativity) could be significantly improved by utilizing service composition in pervasive computing environments. We term this vision as pervasive composition. The capability of providing pervasive composition is named pervasive composability. Pervasive composability is the capability of a system to develop applications by combining existing mobile services. We expect this capability to be realized mainly by the support of peer coordination and service collaboration.

Peer coordination (namely, peer-to-peer collaborations) is achieved by forming a composition of collaborative peers and building a global contract. Peer coordination offers a means by which the rules for collaborating peers can be clearly defined and agreed. Peer coordination is a global activity responsible for seeking appropriate parties to cooperate in a task and achieve a goal.

Service collaboration is a local composition, consisting of services within one service peer. This one service peer administrates all services participating in the service composition. Service collaboration is a local activity responsible for completing a small task by organizing local resources within one peer.

In the context of this chapter, peer coordination and service collaboration present a novel view for explaining the roles of choreography and orchestration in service composition. Peer coordination specifies a global protocol for peer-to-peer interactions and tracks the sequence of messages at a peer level. Peer coordination entails choreography and BPEL's Abstract Process for observable activity description from a global perspective. Service collaboration specifies a local protocol for describing how Web services interact with each other at the message level. Service collaboration is always controlled by one service peer. Service collaboration entails orchestration and BPEL's Executable Process for non-observable activity description from a local perspective. Separating peer coordination from service collaboration provides the possibility to change service collaboration without affecting the peer coordination, and vice versa.

User's everyday behavior (activity) refers to the daily activities of communicating with peers, requesting, receiving, processing and exchanging information with the surroundings for fulfilling a goal. Building service compositions that support everyday activities normally involve goals, task decomposition, peer coordination, and service collaboration. Everyday activity differs from business activity in the sense that exchanging information, peer coordination and service collaboration are considered not from financial, commercial and industrial aspects.

Pervasive computing (i.e. ubiquitous computing) is a post-desktop model of human-computer interaction in which information processing has been thoroughly integrated into everyday objects and activities [37].

A peer is considered as an organization or party which owns various computing resources. A peer providing services is a service peer. A peer group is a way to combine peers and to advertise specific services that are available to group members. Peers can form groups, join groups, and resign from groups.

Pervasive Service Computing is regarded as a Web service-centric solution to support modern human everyday activities with the emphasis of service composition (building applications from services) and controlling the execution of these applications. In a heuristic view, Ontology-Driven Pervasive Service Composition supports goal planning, task decomposition, peer coordination, service collaboration and logic execution.

Goal serves as a stimulus that inspires person to do actions and accomplish something. Activity is motivated by a goal. Task decomposition refers to the whole-part composition structure of a task model. Each subtask represents an activity that could be automated by a service.

Syntactic service composition refers to as user-driven application development by providing the machine-interpretable meta-information that determines what services can do, which peers perform which services, what functionality services provide and how to use services, e.g. prerequisites. Syntactic service composition is based on syntax, in terms of parameters and names matching without providing any additional meta-information that could be interpreted by the system. Such syntactic information is not enough in offering pervasive and context-aware services. Semantic service composition need to be provided for applications to enable automatic peer and service discovery, composition and invocation with additional elements which can be interpreted by the interacting applications so that system is aware of the current situation and context and is able to intellectually reason about the goals and the situation.

Pervasive Service Composition Language (PSCL) provides and defines the minimal set of concepts and essential constructs necessary to specify peer coordination and service collaboration for automating everyday activities. PSCL is based on XML and defined by XML schema.

3 Requirement Analysis

Our previous work examined scenarios created in context of pervasive computing for envisioning future human everyday life and studied a generic user's activity model [40][41]. This section analyzes requirements for Ontology-Driven Pervasive Service Composition.

3.1 Requirements for Pervasive Service Composition

The generic user's activity model abstracts key activities of achieving a goal. In the model, service composition plays an important role in seeking service peers and integrating services. Those two processes are termed in the model as peer coordination and service collaboration. Pervasive Service Composition aims to accommodate peer coordination and service coordination in a pervasive computing environment [40][41].

3.1.1 Requirements for Peer Coordination

Requirements for peer coordination are initially specified as relationship management, interaction management and peer management.

Relationship management establishes what peers do and who can talk with each other within the context of a given task. Within peer coordination, services represent different service peers. A service peer has one of the predefined roles. A relationship is defined for each message exchange between two roles. *Interaction management* is crucial for peer coordination, as it provides the actual logic behind a message exchange between peers. This imposes rules, constraints, and exceptions that must be adhered for an interaction to successfully complete. *Peer management* functionality manages the formation of a peer group, the number of the involving peers, and the joining and leaving of the peers.

3.1.2 Requirements for Service Collaboration

Service collaboration describes centrally controlled workflow logic to automate integrating various services within one peer to achieve the completion of a small everyday task. The workflow logic needs to represent rules, conditions, interactions, and exceptions. The details of such workflow logic are encapsulated and expressed by a service collaboration. Service collaboration requires the ability to define how services within one peer can interact with each other to automate the integration.

3.1.3 Requirements for Service Description

The abstract definition of services in Pervasive Service Composition reuses those elements specified in WSDL [16]:

- *Types*: a container for data type definitions using some type system (such as XSD).
- *Message*: an abstract, typed definition of the data being communicated.
- *Operation*: an abstract description of an action supported by the service.
- *Port Type*: an abstract set of operations supported by one or more endpoints.
- *Binding*: a concrete protocol and data format specification for a particular port type.
- *Port*: a single endpoint defined as a combination of a binding and a network address.
- *Service*: a collection of related endpoints.

3.1.4 Requirements for Interaction Description

An interaction is the basic building block for both peer coordination and service collaboration. Basically an interaction consists of basic activities, exception handling and structured activities for information exchange between peers and services. Activities describe the actions performed within a pervasive service composition.

A basic activity is used to describe the primary actions performed within a pervasive service composition. The potential types of basic activities for service collaboration are: *Invoke* (send a message to a partner service), *Receive* (wait for an incoming message), *Reply* (send a message in reply to a synchronous receive event), *Assign* (copy data between variables), *Empty* (do nothing for modeling the start and end of a composition), *Wait* (wait for a certain period or until a certain deadline), *Throw* (signal a fault) and *Exit* (terminate the execution). Peer coordination has similar types of basic activities, but they are operated between peers is a distributed fashion.

Exception handling defines how exceptional or unusual conditions are handled. We assume that exception handling is needed frequently in Pervasive Service Composition. Several types of exceptions are possible, for example, interaction failures (e.g. the sending of a message does not succeed) and security failures.

Structured activities specify the ordering and conditional relationships between interactions. The potential types of basic activity for both peer coordination and service collaboration are: *sequence* (restrict a set of activities to be performed sequentially in the same order that they are defined), *flow* (execute concurrently), *choice* (specify one of activities to be performed), *pick* (wait for one message or time event), and *while* (repeat while a condition is true).

3.1.5 Requirements for Message Description

Applications in Pervasive Service Composition can involve a large amount of data and information exchange. Such frequent information exchange and data communication occurs in a heterogeneous distributed computing environment within multiple peers. Moreover, in peer-to-peer collaboration there is no central control. Therefore explicit message specification is required for capturing the required information for peer coordination and service collaboration.

3.1.6 Requirements for Context Description

Pervasive Service Composition occurs in a pervasive computing environment. Composed applications have high requirements for adaptability and flexibility. The adaptability raises semantic descriptions of not only computing services, but also computing peers in Pervasive Service Composition.

4 Ontology Model for Pervasive Service Composition

ODPSC (Ontology-Driven Pervasive Service Composition) ontology will be a general ontology of Pervasive Service Composition, for capturing everyday activities including ordinary and business activity. ODPSC ontology will need to be able to address the key requirements given above for achieving a complete service composition as it is typically understood by human beings and parsed and processed by computer. The key concepts needed for ODPSC ontology are depicted in the UML model shown in Fig. 3. The contexts of peer coordination and service collaboration were not detailed at the time of writing this paper. These concepts specify the surroundings of peer coordination and service collaboration such as additional peer and service properties and computing and communication environments. XML Schema data types can be used to develop message ontology and WSDL specification can be used to develop service ontology in ODPSC ontology.

The top concept *composition* contains two major workflow logic concepts, *peer coordination* and *service collaboration*, that make up a plan. Peer coordination

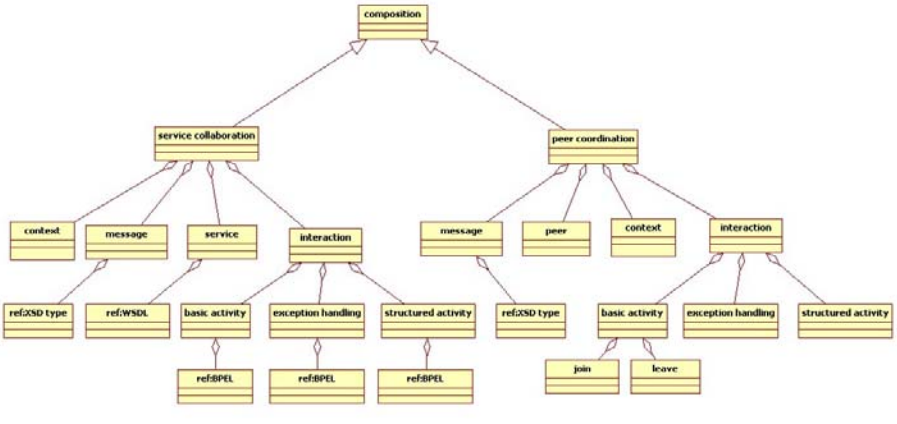


Fig. 3. Ontology model for Pervasive Service Composition

specifies working peers, their roles and relationships, and their interactions for achieving the planned goal in a decentralized and pervasive computing environment. Service collaboration specifies interactions with Web Services within a service peer for an executable process. Basic and structured activity elements defined in BPEL specification can be used to develop service collaboration ontology in ODPSC ontology.

5 Survey of Web service Composition Approaches

This section introduces the existing initiatives of applying ontologies to Web services and service composition with respects to syntactic and semantic service composition. Table 4 lists selected initiatives of service composition. It gives information about the semantic and syntactic ways of service description, how orchestration and choreography is understood within the initiatives and whether peer coordination and service collaboration can be achieved (at least descriptively). An extensive overview of service composition methods can be obtained from [19, 20][21].

Table 4. Summary of Web service composition approaches

Method	Service description		Facilities	
	Semantic	Syntactic	Peer coordination	Service collaboration
WSMO	WSDL, XML Schema	Grounding to WSDL	possible	possible
OWL-S	OWL	Grounding to WSDL	possible but with external facilities involved	possible
SWSF	SWSL	Grounding to WSDL	possible	possible
IRS-III	OCML, XML Schema	Grounding to WSDL	possible	possible
WSDL-S	OWL, UML and any suitable external ontology language	WSDL-S	no real model, external facilities needed	
WS-BPEL		WS-BPEL	Abstract service evolved to overcome with the interactions between parties	possible
WS-CDL		WS-CDL, XPath 1.0	possible	possible

5.1 Syntactic Web Service Composition

This section overviews two syntactic Web service composition approaches, Business Process Execution Language (BPEL) and Choreography Description Language (CDL). These both can be viewed as workflow composition methods.

Business Process Execution Language (BPEL), short for Web Services Business Process Execution Language (WS-BPEL), is a workflow oriented composition approach and has evolved from earlier initiatives, such as XLANG from Microsoft and WSFL (Web Services Flow Language) from IBM. The latest version is WS-BPEL 2.0 [22]. BPEL defines models for describing the behavior of both executable and abstract business processes. An executable process is fully specified and can be executed on the orchestration engine. An abstract process is used to describe the observable message exchange behavior of all involved parties and is hence focused more on the choreography issues. The main elements specified by BPEL are detailed in [22].

Web Service Choreography Description Language (WS-CDL) aims at the composition of interoperable collaborations between any types of participants regardless of the supporting platform or programming model [2]. WS-CDL is an XML based specification for describing such collaboration between participants by defining their observable behavior from a global point of view (e.g. global constraints and ordering rules for information exchanging). Each participant can use this global definition to build and test solutions that conform to it. WS-CDL is not an executable business process description language or implementation language. The main elements specified by WS-CDL are detailed in [2].

5.2 Semantic Web Service Composition

This section overviews approaches to semantic Web service composition focusing on core components and composition principles. The approaches fall into two categories. The approaches in the first category, including WSDL-S, IRS-III and SWSF, extend or use industry standards, while the second category contains new ontologies for semantic Web service composition including OWL-S and WSMO. [23] gives an extensive introduction to these approaches.

Web Service Modeling Ontology (WSMO) is derived from Web Service Modeling Framework [24]. It aims to standardize for semantic Web services a framework that provides support for formal modeling and service representation together with automating service related tasks such as discovery, selection, and composition [25, 26]. The four top-level elements of WSMO are detailed in [24] [27] [28] [29] [30] [31].

Semantic Web Service Ontology OWL-S (formerly DAML-S) is OWL-based ontology that aims to provide a general description of semantic Web services applicable for a wide range of domains to facilitate automatic service discovery, composition, interoperation and invocation [32]. OWL-S consists of three sub-ontologies which are associated with the top-level concept *Service*. These ontologies are ServiceProfile, ServiceModel, and ServiceGrounding [32].

Semantic Web Services Framework (SWSF) [33], proposed by the Semantic Web Services Language Committee, is one of the latest approaches for semantic Web

services. It has two main components: *Semantic Web Service Ontology* (SWSO) and *Semantic Web Service Language* (SWSL). SWSO [34] is a conceptual model for Web service description and formalization. Formalization can be provided by one of the SWSL variants: SWSL-FOL which is based on first-order logic or SWSL-Rules that relies on logic programming. The corresponding ontologies are called FLOWS, First-Order Logic Ontology for Web Services, and ROWS, Rule Ontology for Web Services.

Internet Reasoning Service-III (IRS-III) [4, 35] is a framework for creating and executing semantic Web services. The semantic broker based approach is applied to mediate between service requesters and service providers. IRS-III incorporates the WSMO conceptual model and provides execution environment. Own ontology representation language (OCML) is used to describe interoperability and collaboration from both the client and provider viewpoints. The framework has three main components: ISR-III server, publishing platforms and clients that communicate via SOAP. IRS-III server builds upon a HTTP Server extended with a SOAP handler. The server contains a semantic Web service library storing the semantic descriptions (written in OCML) that are associated with the Web services. The library is structured into domain ontologies and knowledge models for goals, Web services, and mediators. Web services are associated with orchestrations and choreographies.

WSDL-S extends the widely used Web Service Description Language (WSDL) by adding semantics to the functional descriptions of Web services [36]. The main advantages argued by the developers are that WSDL is used to describe both the semantics and operation level details of semantic Web services and that WSDL-S allows Web service developers to annotate Web services with their choice of modeling languages (e.g. UML or OWL). The capabilities and requirements of Web services are annotated with referring concepts from a semantic model. Mechanisms for annotating services, their inputs, outputs, preconditions and effects are provided.

6 Conclusions and Future Work

To define Pervasive Service Composition for everyday life, this paper addressed the fundamental issues of task-based, pervasive, semantic, and P2P-oriented computing in Pervasive Service Composition. Definitions related to service composition, pervasive computing, ontology, and Pervasive Service Composition were given. Consequently, requirements for describing everyday activities were analyzed with respect to peer coordination, service collaboration, service description, interaction description, message description, and context description. To meet those requirements, an initial ontology model was designed for capturing everyday activity and accommodating peer coordination and service collaboration in Pervasive Service Composition. Finally, existing approaches to Web service composition were classified into syntactic and semantics approaches.

We identify three major tasks for the future works as follows. The first task will be to specify a Pervasive Service Composition Language, based on the initial design of the ontology model and utilizing a formal approach like XML. The second task will be to develop a platform for peer coordination. This platform implements a set

of protocols designed to address the establishment of global contract and observable processes between peers. The third task will be to develop a platform for service collaboration.

Acknowledgements

This work was carried out in the Pervasive Service Computing project funded by the Ubiquitous Computing and Diversity of Communication (MOTIVE) program of the Academy of Finland and in the ITEA2-CAM4Home project funded by the Finnish Funding Agency for Technology and Innovation (Tekes).

References

[1] BPEL4WS, Business Process Execution Language for Web Services (2007), http://xml.coverpages.org/WS-BPEL-CS01.pdf (retrieved by May 30, 2010)

[2] Barros, A., Dumas, M., Oaks, P.: A critical overview of the web services choreography description language. BPTrends (March 2005), http://www.bptrends.com/publicationfiles/03-05%20wp%20ws-cdl%20barros%20et%20al.pdf (retrieved by May 30, 2010)

[3] Miller, J., Verma, K., Rajasekaran, P., Sheth, A., Aggarwal, R., Sivashanmugam, K.: WSDL-S: A Proposal to W3C WSDL 2.0 Committee (2004), http://lsdis.cs.uga.edu/projects/wsdl-s/WSDL-S.pdf (retrieved by May 30, 2010)

[4] Domingue, J., Cabral, L., Galizia, S., Tanasescu, V., Gugliotta, A., Norton, B., Pedrinaci, C.: IRS-III: A Broker-based Approach to Semantic Web Services. Journal of Web Semantics 6, 109–132 (2008)

[5] Battle, S., Bernstein, A., Boley, H., Grosof, B., Gruninger, M., Hull, R., Kifer, M., Martin, D., McIlraith, S., McGuinness, D., Su, J., Tabet , S.: Semantic Web Services Framework (SWSF), Overview (September 2005), http://www.w3.org/Submission/SWSF/ (retrieved by May 30, 2010)

[6] Weiser, M.: The computer for the 21st century. Scientific American 265, 94–104 (1991)

[7] da Costa, C.A.: Toward a General Software Infrastructure for Ubiquitous Computing. IEEE Pervasive Computing 7, 64–73 (2008)

[8] Issarny, V., Sacchetti, D., Tartanoglu, F., Sailhan, F., Chibout, R., Levy, N., Talamona, A.: Developing Ambient Intelligence Systems: A Solution based on Web Services. Autom. Software. Eng. 12 (2005)

[9] IST-Amigo, Amigo: Ambient intelligence for the networked home environment (2006), http://www.hitech-projects.com/euprojects/amigo/ (retrieved by May 30, 2010)

[10] Ylianttila, M., Harjula, E., Koskela, T., Sauvola, J.: Analytical Model for Mobile P2P Data Management Systems. In: Proceedings of 5th IEEE on Consumer Communications and Networking Conference, CCNC 2008, pp. 1186–1190 (2008)

[11] Pakkala, D., Koivukoski, A., Paaso, T., Latvakoski, J.: P2P middleware for extending the reach, scale and functionality of content delivery networks. In: Proceedings of Second International Conference on Internet and Web Applications and Services, p. 5 (2007)

[12] p2psip working group, Peer-to-Peer Session Initiation Protocol Specification (2009-03-30), http://www.ietf.org/html.charters/p2psip-charter.html (retreived by May 30, 2010)
[13] Lee, T.B., Hendler, J., Lassila, O.: The Semantic Web. Sci. Am. (May 2001)
[14] Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., Orchard, D.: Web services architecture (February 2004), http://www.w3.org/TR/ws-arch/ (retrieved by May 30, 2010)
[15] W. C. SOAP: SOAP Version 1.2 Part 1: Messaging Framework (2003), http://www.w3.org/TR/2003/REC-soap12-part1-20030624/ (retrieved by May 30, 2010)
[16] W3C-WSDL, WSDL: Web Services Description Language (WSDL) 1.1 (2005), http://www.w3.org/TR/wsdl
[17] UDDI, UDDI Version 3.0.2 (2004), http://www.Oasis-Open.org/committees/uddi-spec/doc/spec/v3/uddi-v3.0.2-20041019.Htm (retrieved by May 30, 2010)
[18] Weerawarana, S., Curbera, F.: Business Process with BPEL4WS: Understanding BPEL4WS (2002), http://www-106.ibm.com/developerworks/webservices/library/ws-bpelcol1/ (retrieved by May 30, 2010)
[19] Cabral, L., Domingue, J., Motta, E., Payne, T., Hakimpour, F.: Approaches to semantic web services: An overview and comparison. In: Bussler, C.J., Davies, J., Fensel, D., Studer, R. (eds.) ESWS 2004. LNCS, vol. 3053, pp. 225–239. Springer, Heidelberg (2004)
[20] Rao, J., Su, X.: A survey on automated web service composition methods. In: Cardoso, J., Sheth, A.P. (eds.) SWSWPC 2004. LNCS, vol. 3387, pp. 43–54. Springer, Heidelberg (2005)
[21] Tabatabaei, S.G., Kadir, W.M., Ibrahim, S.: A comparative evaluation of state-of-the-art approaches for web service composition. In: Proceedings of 2008 the Third international Conference on Software Engineering Advances, Washington, DC, USA, pp. 488–493 (2008)
[22] Jordan, D., Evdemon, J., Chairs, Alves, A., Arkin, A., Askary, S., Barreto, C., Bloch, B., Curbera, F., Ford, M., Goland, Y., Guízar, A., Kartha, N., Kevin Liu, C., Khalaf, R., König, D., Marin, M., Mehta, V., Thatte, S., van der Rijn, D., Yendluri, P., Yiu, A. (eds.): Web Services Business Process Execution Language Version 2.0, OASIS Standard (2007), http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf (retrieved by May 30 , 2010)
[23] Roman, D., de Bruijn, J., Mocan, A., Toma, I., Lausen, H., Kopecký, J., Fensel, D., Domingue, J., Galizia, S., Cabral, L.: Semantic web services - approaches and perspectives. In: Davies, J., Warren, P., Studer, R. (eds.) Semantic Web Technologies: Trends and Research in Ontology-Based Systems. John Wiley & Sons, Chichester (2006)
[24] Fensel, D., Bussler, C.: The Web Service Modeling Framework WSMF. Electronic Commerce: Research and Applications 1, 113–137 (2002)
[25] WSMO working group, WSMO (2009), http://www.wsmo.org (retrieved by May 30, 2010)
[26] Roman, D., Scicluna, J. (eds.): Orchestration in WSMO.WSMO Final Draft (2007), http://www.wsmo.org/2005/d15/v0.1/20070422/ (retrieved by May 30, 2010)

[27] Roman, D., Keller, U., Lausen, H., de Bruijn, J., Lara, R., Stollberg, M., Polleres, A., Feier, C., Bussler, C., Fensel, D.: Web Service Modeling Ontology. Applied Ontology 1, 77–106 (2005)

[28] de Bruijn, J.: The Web service modeling language WSML. WSML Final Draft (2005), http://www.wsmo.org/TR/d16/d16.1/v0.21/ (retrieved by May 30, 2010)

[29] WSMX, Web service execution environment (2009), http://www.wsmx.org/index.html (retrieved by May 30, 2010)

[30] Roman, D., Scicluna, J., Nitzsche, J. (eds.): Ontology-based Choreography. WSMO Final Draft (2007), http://www.wsmo.org/TR/d14/v1.0/ (retrieved by May 30, 2010)

[31] Roman, D. (ed.): Choreography in WSMO. WSMO Working Draft (2004), http://www.wsmo.org/2004/d14/v0.1/20041112/ (retrieved by May 30, 2010)

[32] Martin, D., Burstein, M., Hobbs, J., et al.: OWL-S: Semantic Markup for Web Services (2004), http://www.w3.org/Submission/OWL-S/ (retrieved by May 30, 2010)

[33] Battle, S., Bernstein, A., Boley, H., Grosof, B., Gruninger, M., Hull, R., Kifer, M., Martin, D., McIlraith, S., McGuinness, D., Su, J., Tabet, S.: Semantic Web Services Language (SWSL) (September 2005), http://www.w3.org/Submission/SWSF-SWSL/ (retrieved by May 30, 2010)

[34] Battle, S., Bernstein, A., Boley, H., Grosof, B., Gruninger, M., Hull, R., Kifer, M., Martin, D., McIlraith, S., McGuinness, D., Su, J., Tabet, S.: Semantic Web Services Ontology (SWSO) (September 2005), http://www.w3.org/Submission/SWSF-SWSO/ (retrieved by May 30, 2010)

[35] Cabral, L., Domingue, J., Galizia, S., Gugliotta, A., Norton, B., Tanasescu, V., Pedrinaci, C.: IRS-III: A broker for semantic web services based applications. Journal of Web Semantics 6(2), 109–132 (2008)

[36] Akkiraju, R., Farrell, J., Miller, J., Nagarajan, M., Schmidt, M., Sheth, A., Verma, K.: Web service semantics - WSDL-S. Tech. Rep. A joint UGA-IBM Technical Note, version 1.0 (April 18, 2005), http://www.w3.org/Submission/WSDL-S/ (retrieved by May 30, 2010)

[37] Ubiquitous_computing, Wikipedia, http://en.wikipedia.org/wiki/Ubiquitous_computing (retrieved by May 30, 2010)

[38] Contract, free dictionary, http://www.thefreedictionary.com/contract (retrieved by May 30, 2010)

[39] business activity, http://www.thefreedictionary.com/business+activity (retrieved by May 30, 2010)

[40] Zhou, J., Sun, J., Rautiainen, M., Davidyuk, O., Liu, M., Gilman, E., Su, X., Ylianttila, M., Riekki, J.: PSC-RM: Reference Model for Pervasive Service Composition. In: IEEE Proceedings of Frontier of Computer Science and Technology (FCST 2009), Shanghai, China, December 17-19, pp. 705–709 (2009)

[41] Zhou, J., Riekki, J., Ylianttila, M.: Modeling Service Composition and Exploring its Characteristics, Modeling pervasive service composition. In: IEEE Proceedings of 3rd International Workshop on Web Service Composition and Adaptation (WSCA 2009) in IEEE ICWS 2009, LA, USA, July 6-10, pp. 446–451 (2009)

Navigating the Web of Things: Visualizing and Interacting with Web-Enabled Objects

Mathieu Boussard[1] and Pierrick Thébault[1,2]

[1] Alcatel-Lucent Bell Labs France, Route de Villejust,
91620 Nozay, France
{mathieu.boussard,Pierrick.Thebault}@alcatel-lucent.com
[2] Arts et Métiers Paristech, LAMPA, 2 Bd du Ronceray,
49000 Angers, France
pi.laval@ensam.fr

Abstract. The Web of Things vision aims to turn real-world objects into resources of the Web. The creation of accessible and addressable Virtual Objects, i.e. services that expose the status and capabilities of connected real-world objects using REST APIs, allows for new machine-to-machine interactions to be designed but also for a new user experience to be shaped. Indeed, the change brought about by the connectivity of objects and their ability to share information raises design issues that have to be considered by manufacturers and service providers alike. In this paper, we present an approach to the Web of Things based on both technical and user-centered points of view. We argue that new user interfaces have to be designed to allow users to visualize and interact with virtual objects and environments that host them. We illustrate our vision with an interaction mockup that allows the user to navigate the Web of Things through three devices (a smart phone, a web tablet and a desktop web browser).

Keywords: Pervasive computing, Internet of Things, Web of Things, user experience, interaction design, user interfaces, ambient intelligence, browser.

1 Introduction

When the term 'Ubiquitous computing' was coined in the early 90's by Mark Weiser from Xerox PARC [1], it was a mere vision depicting information services processed and consumed outside of the personal computer terminal. Years have passed and consequent research work has been conducted in this field trying to address subsets of the bigger problem, but it was not before the past few years that conditions to actually embody this vision in people's everyday life were seemingly met.

A first phenomenon is the advent of networking as a basic (hardware) feature – it has become cheap enough in the last few years to embed networking capabilities in every object, starting with powerful mobile computing devices but also for everyday objects with limited computation or interface capabilities. In parallel, consequent research was conducted under the umbrella of the "Internet of Things" on how to actually network these objects based on a common networking technology: IP. Finally, the World Wide Web as imposed itself as the main platform to deliver services

T. Margaria and B. Steffen (Eds.): ISoLA 2010, Part I, LNCS 6415, pp. 390–398, 2010.

to end-user, thanks to its simple and open foundations that fostered service creation by virtually any of its users, in all aspects of people's lives.

As a result, it has now become possible to embody the ubiquitous computing paradigm using a *Web of Things* approach – where web technology allows exposing real-world objects as resources on the Web. As a consequence, users can interact with objects via a virtual representation of objects that surround them. In this paper we discuss how user experience should be considered as crucial in the design and use of connected objects.

In chapter II, we discuss the overall concepts underlying the Web of Things and related work. We then present in chapter III general considerations regarding user experience in environments where objects are exposed as Web resources, before presenting some early results in chapter IV. We finally conclude and present future opportunities in chapter V.

2 An approach to the Web of Things

Just as the term "Internet of Things" covers different aspects ranging from object identification to wireless sensor networking, the expression Web of Things can be used to describe many different topics. These range from embedding a Web server in constrained resource devices to exposing real-world (connected) objects using Web techniques and composing them. In the following, after presenting related work, we focus on aspects that have a direct impact on user experience in a Web of Things enabled environment.

2.1 Related Work

Approaches to apply web principles to ubiquitous environments originate from the early 2000s when the Web gained popularity as a delivery platform. Kindberg et al. discussed the concept of "Web presence" under the form of a web page for people, places and things in [2]. A number of solutions using web-related standards and methods were applied, from distributed computing middleware like JXTA [3] to Web Services technologies and for example the device-oriented flavor of the protocol suite DPWS [4]. Using such solutions requires specific skills from the service developer to create an application that takes full advantage of it, and potentially specific software to be installed.

The term "Web of Things" was coined recently as an application-oriented reaction to the network-oriented Internet of Things research, also feeding on the Web 2.0 concept of the Web as a (service-)platform [5]. Initiatives like SenseWeb and Pachube have focused on a data-oriented exposure of real world objects, where a centralized Web service proposes a repository for sensor readings, exposing an API for third-party mashups ([6], [7]). Initiatives providing service-oriented exposure to objects themselves seem more interesting to us, as they are likely to provide greater scalability and interaction possibilities. Such approaches are described in [8] and [9] where the advantages of using of RESTful principles to expose real-world connected objects to Web applications are discussed.

Although a great deal has been achieved, most of this work has been focusing on machine-to-machine (M2M) aspects, and very little on the representation to users of possibilities offered by Web-exposed objects. Most work relies on HTML pages to represent and interact with networked objects. Although as stated in [2] the browser is the de-facto interface to the Web and has been broadly accepted by end-users, recent developments on HTML5, Rich Internet Applications, and mobile interfaces have shown that there is still a lot of room for improvement presentation-wise.

2.2 Virtual Objects

In our approach, we focus on exposing connected real-world objects (RWO) as resources accessible using Web technology – we call such avatars of physical objects *Virtual Objects* (VO). Virtual Objects are responsible for exposing the RWO state and functionalities on the Web; they should be addressable (via a URI), accessible (presenting a REST interface) and provide a description of their nature, status and capabilities towards both other services (for M2M purposes) and users through a visual representation.

Depending on the nature of the RWO, different realization of the VO and related hosting schemes have to be considered. If the RWO is Web-enabled, its VO can be embedded in device. In another schema, a gateway could host a collection of VOs based on network topology (e.g. of all objects connected to a network access point) or geographical considerations (e.g. all devices in a building or room). This solution is interesting in particular for objects that are not Web- or even IP- enabled, comparable to the work described in [8]. Other deployment schemas can be envisioned, such as the hosting in a cloud providers' solution. It is worth noting that deployment schemes are not only bound by technical choices – one could imagine different business models where the manufacturer of an object would rather host the related VO on behalf of their customers, or such hosting service being provided by the user's Internet Service Provider, etc.

As Virtual Objects are the finest grain of interaction users could meet in Web of Things experience, it is important that they provide a representation to users, in a way that is best suited to the usage situation. This representation should provide the user with the possibility to interact with services exposed by objects (e.g. turn lamp on/off) and to monitor user-relevant object state (lamp is on).

2.3 Composing Objects of an Environment and Using Them in Applications

As pointed out in [3], "while the Web is a global-scale information system for which physical location is transparent, we often want to access resources that are conveniently near our physical location". It is therefore reasonable to think that although searching for virtual objects representing remote real-world objects will have its utility, visualizing objects *in-situation* (i.e. those present in the current environment of the user) is crucial. For this, it is necessary to model the concept of environment (as a grouping of physically regrouped objects, but also of potential compositions between these objects). In our approach we map the concept of environment on a virtual object gateway which hosts the VOs of the objects connected to a same network access point, and as a consequence are located in the same premises.

As objects get exposed as services on the Web, they enable new types of composition or *mashups*. Guinard et al. [8] propose a first classification using the terms *virtual-physical mashups* for applications that present a web user interface to objects and *physical-physical mashups* for M2M compositions. While the latter are directly constrained by the nature and capabilities of the composed objects (e.g. the composition of a webcam with a display is directly linked to the nature of the two considered objects), virtual-physical mashups can cover a much broader range of application logic, only limited by the service developer imagination or needs. It is important to note that from a user perspective both approaches should be made visible as they participate to the ambient intelligence of an environment.

3 Towards New User Experience

As the exposition of real-world objects on the Web allows the design of machine-to-machine interactions, objects can be considered as actuators putting specific physical or digital mechanisms in action, without any immediate user input. This leads to the creation of ambient intelligence that is able to control object behaviors. Alternative interfaces designed for a new means of control of objects allow for a new experience to be shaped by manufacturers, service providers or end-users in order to assist people in their daily life. From a user-centered perspective, we propose to discuss, in the following sections, the issues related to the perception of objects' connectivity by users, the representation of interactions between objects and the creation of object groups.

3.1 Distinguishing Web-Enabled Objects from Non-connected Objects

Users are likely to experience automation of multiple objects working together to relieve the effort of turning on/off and configuring their devices or appliances. The communication of two or more objects through the Web is initiated without explicit permission from users and remains invisible to them. As communication chipsets are often hidden inside objects, users will not make, in most cases, the difference between connected and non-connected devices. Norman [10] tells us that user observation of feedback depends upon the information conveyed by the physical device itself. We argue that new graphic representations are needed to make the Internet connection and VOs more tangible and perceptible by users. Many different solutions ranging from a simple marker to a complete redesign of object's user interfaces can be proposed. As Arnall [11] introduced a graphic language for touch-based interactions, we need to explore the visual link between information and real-world object. Regardless of whether Internet capability of objects is made visible, users are likely to be surprised by the automation of their surrounding, especially if they have not shaped the objects' behaviors by themselves. Woods tells us that situations where "automated systems act in some way outside the expectations of their human supervisor" [12] are caused by a poor feedback about the activities of automated systems to users. "Automation surprises" and difficulties to anticipate objects activities can be in some ways avoided by the integration of a dynamic visual feedback, for example a LED that indicates to users a possible automation. We argue that a means of controlling the

Internet connection status is needed in a transitional phase where users can perceive objects' intelligence as a form of manipulation. It is crucial that objects' automation does not jeopardize the collective use of products or constitute a barrier to the acceptance of the Web of Things vision.

3.2 Understanding Objects' Behaviors

New exchange capabilities of objects allows for the design of behaviors to be shaped according to their features' descriptions. Virtual links between two objects can be drawn by users and considered as a representation of interoperable functions. From a user perspective, it seems crucial to be able to figure out the complexity of this network. It is therefore needed to specify the link direction, related to the input/output relationship of devices, to help people understand how objects interact together. If such associations can, to some extent, be displayed on devices equipped with screens (e.g. phone, television, radio, etc.), it would be impossible to do so with traditional appliances (e.g. lamp, heater, shutter, etc.). We argue that consistent and flexible visualization modes are needed to provide a broader and quick understanding of object's behaviors. If it remains difficult for the user to comprehend how several objects spread in different rooms or locations are working, it is also harder to perceive the unification of several objects by a web application hosted on the cloud. Such applications make possible complex behavior modeling and allow the user to script customized events triggered by an ambient intelligence. The application therefore operates the functions of objects even if the input/output characteristics cannot be combined. This leads to the creation of composite applications using multiple objects that have to be presented to the user in a comprehensible way.

3.3 Grouping Objects

The capacity to control objects from a remote location or system raises issues related to privacy and ownership. To avoid misuse, it is needed to ensure that people have the appropriate rights to access virtual objects and make modifications to their behaviors. It is however extremely difficult for the system to determine what can be done or not with objects. Insofar an object can be offered by a user to another, lent to somebody else or sold for second-hand use, it would require the users to constantly update the access rights of their objects on dedicated applications. We argue that it is possible to identify users through the devices used to browse virtual objects and to build a social network of "things" built on top of existing platforms.

Sterling [13] tells us that next generation of objects called "spimes" will be tracked through space and time throughout its lifetime. The environment of use is therefore another parameter that has to be considered even if GPS chipsets do not provide relevant measures to locate an object in indoor environments [14]. Virtual objects can however be attached to a geographically described environment. As objects are most likely to be wirelessly connected to residential set top boxes, the latter, or a dedicated access point, can be considered as a geographical markup and provide a global understanding of the environment. This allows the creation of abstract environments bridging multiple physical places (e.g. a campus of several building in different neighbors) that users are likely to visit in a specific context.

4 Illustration of the Web of Things Experience

In order to illustrate our Web of Things vision, we designed an interaction mockup that allows users to interact with connected objects (including a lamp, a webcam, a computer screen and a fixed phone) in a real life-like context. We argue that a new user experience can be shaped by offering interactions with terminals that makes up for the lack of dedicated user interfaces on objects. This approach led us to offer the users three interfaces designed for a smart phone, a touch tablet and a desktop web browser. We discuss below that each of these applications is best suited for specific tasks and context (in-situation or off-situation), as described in the following sections and summarized in Table 1.

Table 1. Characteristics of the three designed interfaces

	Smart phone	Touch Tablet	Web browser
Browse from	One object	An environment	All objects, all environments
Actions	Quick visualization and authoring	Visualization, authoring, application download	Advanced visualization, authoring, application download and coding
Visualization	Augmented reality	Graph, lists	Graph, lists, maps, search results
Interactions	Shorts interactions	Short and long interactions	Long interactions
Posture	Stand up	Relaxed	Desk
Use	Personal	Semi-personal (with authentication)	Shared (with authentication)

4.1 Browsing Virtual Objects In-Situation

As most of objects are generally used in-situation, users are likely to access virtual objects while being close to the related real-world object. To provide a quick overview of the objects' properties and active links, we rely on mobile phones and web tablets. Both terminals can be used to deliver a specific experience based on observed user's practices.

Mobile. The Internet capabilities of smart phones and the integration of camera, GPS or RFID reader allow service providers to design new interactions in public space [15]. Users have learnt that barcodes and matrix codes are encoded information or links that can be scanned using specific applications in order to access content more easily. In the same way augmented reality technologies are used to superpose digital representation on reality [16], we created a mobile phone application that allows the user to "reveal" the capabilities and behavior of Web-enabled objects (see Figure 1). This is made possible by directly shooting the camera at objects equipped with visual markers. The latter therefore bridge the virtual and physical objects and can be considered as pointer to the corresponding Virtual Object. The aim is not only to provide a quick understanding of an object's behavior, but also interactions with the Virtual

Fig. 1. Snapshots of the three access modalities: mobile, tablet and desktop browser

Object. We argue that users are likely to want to deactivate a link or quickly setup a new one and need quick access modes and immediate feedback. However, screen sizes of mobile phones restrict the possibilities and require us to develop short interactions that can be handled while a user is standing up in front of an object.

Tablet. Web or media tablets aim to fill the gap between the desktop computer and the mobile phone. If the latter is also used as an Internet access terminal in homes, touch tablets offer more natural interactions with the web and bigger screens [17, 18]. Whereas the mobile phone allows contextualizing the information of an object through the mechanism described above, we argue that tablets are more appropriate to give an overview an environment. McClard and Somers [19] tell us that such devices allow multitasking, relaxed posture of use and play a social role in family. We therefore designed an application that can be considered as a "radar" providing an abstract representation of the user environment and its Web-enabled objects. It allows the users to browse the Virtual Objects corresponding to objects available in the immediate space, through multiple modalities (e.g. a graph with icons, lists, search engine, bookmarks), and explore distant known environments (e.g. grandma's house, office, etc.). We argue that users are likely to want to administrate their ecosystem, create new links and applications and browse existing applications with the tablet. As the device is meant to be shared with different users, it is needed to ask for authentication to activate authoring modes.

4.2 Browsing Virtual Objects Off-Situation

If the mobile and tablet interfaces provide a new means of visualization and control over Web-enabled objects, an online aggregator specifically designed for desktop and laptop computers is needed for longer and more complex interactions with Virtual Objects. Personal computers are widely use to organize, maintain or retrieve information and can be considered as an essential asset to other device's management (e.g. a PC is needed to transfer files from and to a mp3 player, camera or smart phone). We argue that users are likely to want to browse all their virtual objects through a web dashboard providing multiple access modalities (e.g. graphs, lists, maps, galleries) and a robust search engine. As mentioned in part 2, there is an opportunity to build a new social platform allowing users to navigate through objects, environment and people. Whereas websites like ThingD [20] and ThingLink [21], Talesofthings [22],

Stickybits [23] offer to bookmark and comment a catalog of objects, we propose to rely on existing social graph (e.g. Twitter, Facebook, FlickR) to add a social dimension to objects. The desktop application can also offer authoring tools to create object-based applications and mashups. Specific solutions can finally be designed by manufacturers or building administrators to monitor and control physical assets.

5 Conclusion

In this paper we have discussed how exposing real-world objects as resources on the Web enables to realize the old pervasive computing paradigm. In our approach of this Web of Things, we use the concept of Virtual Object to represent the real-world object as a service to other Web entities, and enable compositions that participate in ambient intelligence. We argue that to ensure the success of such a vision, user experience should be carefully designed, in order to make users aware of the possibilities and behaviors in the environment they visit. To illustrate these considerations, we developed three user interfaces allowing users to interact with connected objects of an environment, using mobile interfaces for in-situation browsing. Further work includes actual user testing of both these interfaces and the underlying principles. By mixing social and location-based approaches, we argue that a suitable user experience can be shaped to offer users a new means of navigating the Web of Things - keeping in mind that too much participation from the user or complex representations of their personal ecosystems will lead to a poor adoption of the vision.

References

1. Weiser, M.: The computer for the 21st century. Scientific American 272, 78–89 (1995)
2. Kindberg, T., Barton, J., Morgan, J., Becker, G., Caswell, D., Debaty, P., Gopal, G., Frid, M., Krishnan, V., Morris, H.: others: People, places, things: Web presence for the real world. Mobile Networks and Applications 7, 365–376 (2002)
3. Traversat, B., Abdelaziz, M., Doolin, D., Duigou, M., Hugly, J.C., Pouyoul, E.: Project JXTA-C: Enabling a web of things. In: Proceedings of the 36th Annual Hawaii International Conference on System Sciences, p. 9 (2003)
4. Chan, S., Conti, D., Kaler, C., Kuehnel, T., Regnier, A., Roe, B., Sather, D., Schlimmer, J., Sekine, H., Thelin, J.: others: Devices profile for web services. Microsoft Developers Network Library (May 2005)
5. What Is Web 2.0, `http://oreilly.com/web2/archive/what-is-web-20.html`
6. Kansal, A., Nath, S., Liu, J., Zhao, F.: Senseweb: An infrastructure for shared sensing. IEEE multimedia 14, 8 (2007)
7. pachube: connecting environments, patching the planet, `http://www.pachube.com/`
8. Guinard, D., Trifa, V.: Towards the web of things: Web mashups for embedded devices. In: Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web (MEM 2009), in Proceedings of WWW (International World Wide Web Conferences), Madrid, Spain (2009)
9. Wilde, E.: Putting things to REST. School of Information, UC Berkeley, Tech. Rep. UCB iSchool Report. 15 (2007)
10. Norman, D.A.: The design of everyday things. Basic Books, New York (2002)

11. Arnall, T.: A graphic language for touch-based interactions. In: Proceedings of Mobile Interaction with the Real World MIRW (2006)
12. Woods, D.D.: Decomposing automation: Apparent simplicity, real complexity. In: Automation and human performance: Theory and applications, pp. 3–17 (1996)
13. Sterling, B., Wild, L.: Shaping things. MIT Press, Cambridge (2005)
14. Chen, G., Kotz, D.: A survey of context-aware mobile computing research, Citeseer (2000)
15. O'Reilly, T., Battelle, J.: Web squared: Web 2.0 five years on. In: Proc. of the 6th Annual Web 2
16. Azuma, R.T.: others: A survey of augmented reality. Presence-Teleoperators and Virtual Environments 6, 355–385 (1997)
17. Saffer, D.: Designing Gestural Interfaces. O'Reilly Media, Inc., Sebastopol (2008)
18. Valli, A.: Natural Interaction White Paper
19. McClard, A., Somers, P.: Unleashed: Web tablet integration into the home. In: Proceedings of the SIGCHI conference on Human factors in computing systems, pp. 1–8 (2000)
20. thingd, `http://www.thingd.com/`
21. Thinglink, `http://www.thinglink.org/`weSwitch
22. Tales of Things, `http://www.talesofthings.com/`
23. stickybits - tag your worldTM, `http://www.stickybits.com/`

Shaping Future Service Environments with the Cloud and Internet of Things: Networking Challenges and Service Evolution

Gyu Myoung Lee and Noel Crespi

Institut Telecom, Telecom SudParis
9 rue Charles Fourier, 91011, Evry Cedex, France
{gm.lee,noel.crespi}@it-sudparis.eu

Abstract. To address the new paradigm of future services, cloud computing will be essential for integrating storage and computing functions with the network. As many new types of devices will be connected to networks in the future, it is very important to provide ubiquitous networking capabilities for "connecting to anything" between humans and objects for realize the Internet of Things (IoT). This paper introduces several challenges for the cloud computing in telecom perspectives and ubiquitous networking capabilities to support the IoT. For this, we present the basic concepts and present our vision related to this topic. In addition, we clearly identify characteristics and additional capabilities to support key technologies to be used for the IoT. For various services using ubiquitous networking of IoT, we propose the cloud-based IoT which aims to efficiently support varies services using cloud technology from different kinds of objects (e.g., devices, machines, etc). We also emphasize the necessity of virtualization for service evolution using smart environment of the cloud and the IoT.

Keywords: Cloud computing, Internet of things, ubiquitous networking, future Internet.

1 Introduction

New paradigms for future mobile and ubiquitous environments imply decisions regarding the direction for the evolution of networks as well as investigating technologies that will allow an efficient support of new services by the future Internet.

Most of network providers already support basic offers such as simple access to services (e.g., Internet access) based on user devices such as personal computer (PC). Users also expect to access future Internet value-added services, which enhance quality of life and of work. Ubiquitous service capabilities as well as network-based utility monitoring and billing are examples of such value added services. Accordingly, simple and basic broadband "access" oriented business will shift to future Internet-based business opportunities.

The following represents key features of evolving future Internet business-driven services:

T. Margaria and B. Steffen (Eds.): ISoLA 2010, Part I, LNCS 6415, pp. 399–410, 2010.

- **Ubiquity:** for providing anywhere/anytime service with "connecting to anything" feature, e.g., seamless mobility between heterogeneous networks using convergence devices;
- **Personalization:** for personalizing features of application and services;
- **Handy access:** for easing access to services through various terminals using easy, simple, intuitive and consistent user interface(s);
- **Intelligence:** for providing convenient services with automatic recognition and recommending of user's interests and preferences;
- **Broadband:** for delivering multimedia information including data with large traffic volume due to increase of connected devices and increase of bandwidth required by services and applications;
- **Convergence:** for offering services in an integrated way that include fixed, mobile;
- **Quality:** for providing customizable quality of services (QoS)/ quality of experience (QoE) from end-to-end across different provider networks.

Based on network evolutions, future Internet needs to support the architectural principles of both vertical (from transport to services/applications) and horizontal (one end-user to other end-user through user to network and network to network interfaces) perspectives [1]. To cope with new paradigms future services, integrating the network with the storage and computing functions is most critical to the cloud. The telecom providers can leverage their natural advantages more by integrating the network with storage and computing [2].

Looking at the vertical perspective, studies are required in the area of networking capabilities for the control and operation of various multimedia services over complex stacks involving different layer technologies. From a horizontal perspective, further enhancements in the area of user-centric communication capabilities should take into account complex user situations including various devices connected to home networks and various access technologies which support convergence [3]. For so-called the Internet of Things (IoT) [4], these capabilities are necessary to support ubiquitous networking and to provide interconnection between humans and objects, i.e., providing for Any Time, Any Where, Any Service, Any Network and Any Object.

In this paper, we introduce several issues for the cloud computing in telecom perspectives and ubiquitous networking capabilities to support the IoT. For this, we present the basic concepts and expose our vision related to this topic. In addition, we clearly identify characteristics and additional capabilities to support key technologies to be used for the IoT. For various services using ubiquitous networking of IoT, we propose the cloud-based IoT which aims to efficiently support varies services using cloud technology from different kinds of objects (e.g., devices, machines). We also emphasize the necessity of virtualization for service evolution using smart environment of the cloud and the IoT.

The remainder of the paper is organized as follows. In Section 2, we explain characteristics of cloud computing in telecom perspectives. The Section 3 explains the concept and visions of ubiquitous networking for IoT. Then, in Section 4, we propose the cloud-based IoT service environment. In Section 5, we discuss key characteristics, enhanced capabilities for ubiquitous networking as future networking challenges and we present service evolution using smart environment of the cloud and the IoT. Finally in Section 6, we summarize and discuss future work.

2 Cloud Computing in the Telecom Perspectives

In this section, we introduce the concept and characteristics and deployment models of cloud computing. In addition, we investigate several benefits of offering cloud computing services from the telecom perspectives.

The term "cloud" is used as a metaphor for the Internet, based on the cloud drawing used in the past to represent the telephone network, and later to depict the Internet in computer network diagram as an abstraction of the underlying infrastructure it represents. The term "cloud computing" is a used to describe a new class of network based computing that takes place over the Internet, basically a step on from utility computing [5].

From national institute of standards and technology (NIST) [6], cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model promotes availability and is composed of five essential characteristics, three service models, and four deployment models as shown in Figure 1.

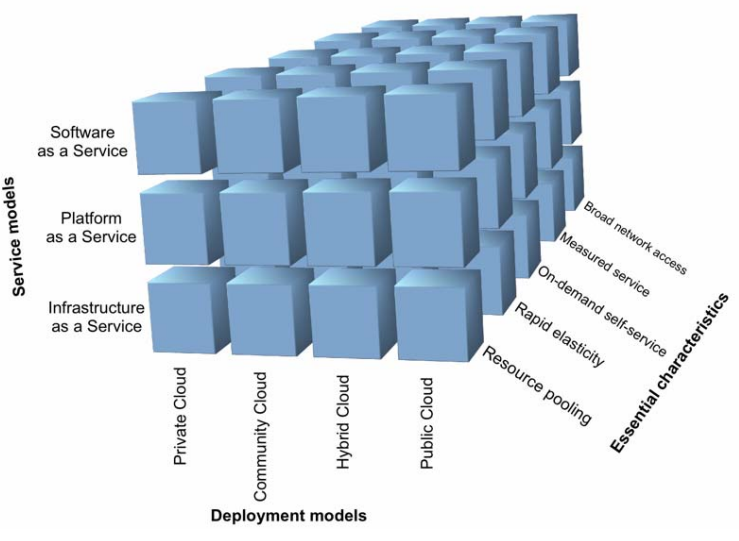


Fig. 1. The service model, deployment models and essential characteristics of cloud computing (illustration from [7])

Virtualization, grid computing, web 2.0, service oriented architecture (SOA), web oriented architecture (WOA), etc., are the technology trends that will, for now, fuel the cloud computing initiative, but these are ephemerons, and the same concept remains regardless of technology changes.

Telecom service providers consider alternative delivery models to acquire and deliver information technology (IT) services demanded by their customers. Service

providers regard their networks as a strategic asset capable of driving incremental revenue and increased profitability in replacement of existing revenue schemes. With a cloud computing services model, service providers can insert themselves into the value chain by redefining their roles to expand beyond connectivity and provide Web-based application delivery services.

There are several reasons why service providers should capitalize on cloud computing for their business and for their customers [8]:

- **Reduced cost**

Cloud technology has the potential impact to minimize operational costs by reducing the hardware and software requirements as well as management costs compared to current networks and platforms.

- **Web-based applications**

Web-based services and applications are suitable for the rapidly changing enterprise workplace. Service providers can increase their revenue and market share and capitalize on Web-based application services by communicating and promoting the tangible business perspectives to their customers.

- **Cloud-based managed services**

Cloud technology offers service providers an ideal model for developing managed services because they already have the scalable engine to build mass services. By assuming an end-to-end position (i.e., application to end user) in the cloud computing value chain, the service provider can improve and add significant quality of service to user-to-application experiences.

- **Carriers' data center efficiency and operations**

A cloud computing data center model enables rapid innovation, scalability and support of core enterprise functions, resulting in significant economies of scale. A cloud computing data center reduces the need for additional hardware, software and facilities, as well as automation of server, network, storage, operating systems and middleware provisioning, and security issues, all of which are costly and time-consuming functions.

- **Differentiating service providers from the pack**

The current economic climate has forced service providers to take a hard look at their business models and how they differentiate themselves from their competitors. Delivering cloud-based consumer and business-critical applications with solid service-level agreements (SLAs) will not only allow service providers to differentiate themselves but will maximize the value of the network while promoting a new business model.

3 Ubiquitous Networking and Vision for the Internet of Things

For IoT, it is critical to extend current networking capabilities to devices/machines for ubiquitous access to the network. For this, we explain the concept and features of ubiquitous networking and also provide vision of the IoT for interdisciplinary fusion revolution crosses over industries.

3.1 Ubiquitous Networking for the Internet of Things

In this paper, we focus on "ubiquitous" perspective from the point of view of networking aspects of the IoT. In this context, the term "ubiquitous networking" is used for naming the networking capabilities which are needed to provide various classes of applications/services which require "Any Services, Any Time, Any Where and Any Objects" type of operation [9].

Figure 2 makes a distinction between the following users of ubiquitous networking: humans (using attached devices such as PC, mobile phones) and objects (such as remote monitoring and information devices, contents).

As shown in Figure 2, ubiquitous networking supports three types of communications:

- **Human-to-Human Communication**: humans communicate with each other using attached devices;
- **Human-to-Object Communication**: humans communicate with a device in order to get specific information (e.g., IPTV content, file transfer);
- **Object-to-Object Communication**: an object delivers information (e.g., sensor related information) to another object with or without involvement of humans.

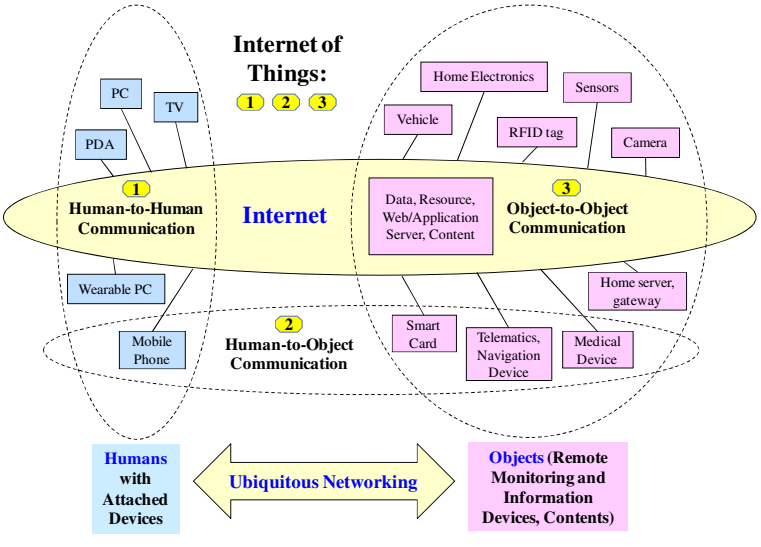


Fig. 2. Ubiquitous networking for IoT – three types of communications

Ubiquitous networking for IoT aims to provide seamless communications between humans, between objects as well as between humans and objects while they move from one location to another.

The capabilities required for the support of ubiquitous networking for IoT are built upon capabilities of current Internet with necessary extensions and/or modifications

of capabilities required for the support of ubiquitous networking services and communications.

Ubiquitous networking in future Internet will support many types of devices connected to the networks. Smart objects such as radio frequency identifier (RFID) tag, sensors, smart cards, medical devices, navigation devices, vehicles as well as the existing personal devices such as PC and smartphones are examples of these [5]. We consider that the end points that are not always humans but may be objects such as devices/machines, and then expanding to small objects and parts of objects. The ubiquitous networking aims to provide "seamless connection between humans, objects and both" while they move from one location to another in pervasive computing environments. Figure 1 shows the extension of the networking domain to support new ubiquitous devices and databases, Web, application servers.

3.2 Interdisciplinary Fusion Revolution Crosses over Industries

This section provides further information regarding the potential directions for network evolution and a vision of ubiquitous networking services, applications and capabilities.

One of the ultimate objectives of ubiquitous networking is to meet the challenge of seamless communications of "anything" (e.g., humans and objects). Ubiquitous networking will have to encompass the following:

- **Ubiquitous connectivity** allowing for whenever, whoever, wherever, whatever types of communications;
- **Pervasive reality** for effective interface to provide connectable real world environments;
- **Ambient intelligence** allowing for innovative communications and providing increased value creation.

As a result, ubiquitous networking will also enable innovative services involving the use of technologies such as bio-technologies (BT), nano-technologies (NT) and content technologies (CT), thus allowing the provision of services that go beyond traditional telecommunication and IT services. These innovative services will require extensions in terms of networking capabilities as well as the access of any type of object.

New businesses using ubiquitous networking require multiple technologies to operate together such as RFID/sensors, protocols, security, and data processing. In order to communicate with related technical parties accommodated in new business relationships, one of the most urgent needs consists in the integration and combination of technologies such as BT, NT or CT. In particular attention needs to be paid to "interdisciplinary fusion" technologies which combine BT, NT, CT as well as IT using ubiquitous networking capabilities. Thus, integrated engineering for new "Interdisciplinary Fusion Revolution" will emerge allowing for extension of services to other industries beyond the IT industry and constituting the vision of ubiquitous networking.

Communication networks have been mainly supporting the evolution of information processing and service capabilities within IT industries. However, the capabilities of networks benefiting from ubiquitous networking should impact other industries

such as medical industry, education industry, finance industry or transportation/distribution industry resulting in new requirements for medical or education networks and services taking into consideration of IT technologies. There are several examples of interdisciplinary fusion services using ubiquitous networking: remote medical services, Intelligent Transport Systems, Supply Chain Management, U-Building or U-City. Providing "fusion services" in future Internet will require that the following capabilities be supported: location tracking, sensing, surveillance and management capabilities.

Businesses using ubiquitous networking will impact on many other industries. Thus, technologies related to architectural functions and enhanced capabilities for the support interdisciplinary fusion services using ubiquitous networking capabilities need to be developed once the basic concept and principles will be ready. Case studies for each service area are also required for helping future developments of emerging Internet technologies.

4 The Cloud-Based Internet of Things

In this section, we introduce evolutional steps of Internet services considering the cloud computing and the IoT. In addition, we propose a new service environment which combines both the cloud computing and the IoT.

Long time ago, we had used stand alone computers (i.e., 1st phase in Figure 3) which contain applications and data. At this time, we didn't need any communication network. With the help of networks, we started to share data from web sites (i.e., 2nd phase in Figure 3). However, the emergence of new computing technologies such as

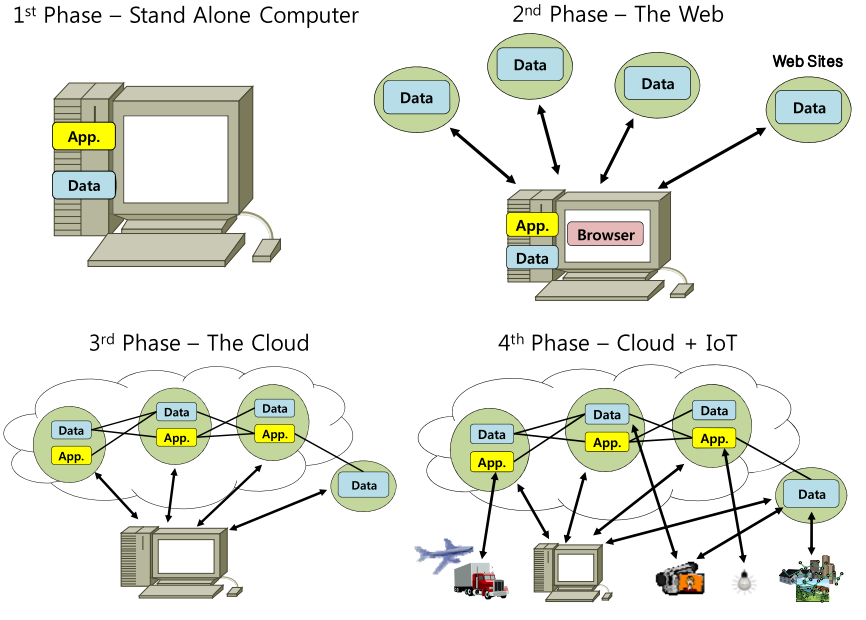


Fig. 3. From stand alone computer to cloud-based IoT

cloud computing is changing the current service paradigms. In case of the cloud (i.e., 3rd phase in Figure 3), hosts such as computers can use resources in cloud which contains data and applications. In the next phase (i.e., 4th phase in Figure 3), cloud computing and IoT will be combined in order to support so many heterogamous objects. These objects are directly attached to the cloud for storing and retrieving of data.

In this paper, we propose a new service provisioning environment – the cloud-based IoT which combines the cloud and the IoT as shown in the 4th phase of Figure 3. The proposed service environment aims to efficiently support various services using cloud technology from different kinds of objects.

There are many advantages for the proposed the cloud-based IoT. These advantages might come from characteristics of cloud computing depending on specific use cases of the IoT. For the cloud-based IoT, we can consider the following points: flexibility of resource allocation, more intelligent applications, energy saving, heterogeneity of smart environment, scalability/agility, virtualization, security, etc.

5 Future Networking Challenges and Service Evolution to Support the Internet of Things

In this section, we provide key characteristics for ubiquitous networking to support the IoT and enhanced capabilities for ubiquitous networking in the IoT environment.

5.1 Characteristics of Ubiquitous Networking for IoT

Fundamental characteristics of ubiquitous networking for IoT are as follows:

- **IP connectivity:**

IP connectivity will allow objects involved in ubiquitous networking to communicate with each other within a network and/or when objects have to be reachable from outside their network. Particularly, as many new types of objects will be connected to networks, IPv6 will play a key role in object-to-object communications using auto-configuration and also mitigate the foreseeable IPv4 address exhaustion.

- **Personalization:**

Personalization will allow to meet the user's needs and to improve the user's service experience since delivering appropriate contents and services to the user. User satisfaction is motivated by the recognition that a user has needs, and meeting them successfully is likely to lead to a satisfying client-customer relationship and re-use of the services offered.

- **Intelligence:**

Numerous network requirements in terms of data handing and processing capabilities will emerge from various industries involved in the field of ubiquitous networking (e.g., the car industry, semi-conductor industry or medical industry). Making these capabilities available for use by business and assisting this business in terms of efficient and timing decision making is very important. Intelligence which enables network capabilities to provide user-centric and context-aware service is therefore essential. Introduction of artificial intelligence techniques in networks will help to accelerate the synergies and ultimately the "fusion" between the involved industries.

- **Tagging objects:**

RFID is one of tag-based solutions for enabling real-time identification and tracking of objects. Tag-based solutions on ubiquitous environment will allow to get and retrieve information of objects from anywhere through the network. As active tags have networking capabilities, a large number of tags will need network addresses for communications. As IP technology will be used for ubiquitous networking, it is essential to develop mapping solutions between tag-based objects (e.g., RFIDs) and IP addresses.

- **Smart devices:**

Smart devices attached to networks can support multiple functions including camera, video recorder, phone, TV, music player. Sensor devices which enable detection of environmental status and sensory information can utilize networking functionalities to enable interconnection between very small devices, so-called 'smart dusts'. Specific environments such as homes, vehicles, buildings will also require adaptive smart devices.

5.2 Enhanced Capabilities for Ubiquitous Networking in the Internet of Things Smart Environment

To establish a set of common principles and architectures for the convergence and ubiquitous environment, enhanced architectural frameworks for IoT are required to facilitate innovation in the use and application of industry capabilities. To cope with changes of future Internet environment, we should take appropriate measures to accommodate the increase in the number of devices.

The high-level capabilities for the support of ubiquitous networking in the IoT smart environment are listed as follows:

- **"Connecting to anything" capabilities**

The capabilities of "connecting to anything" refer to the support of the different ubiquitous networking communication types as described in Section 3.1 and include the support of tag-based devices and sensor devices. Identification, naming, and addressing capabilities are essential for supporting "connecting to anything" [10].

- **Open web-based service environment capabilities**

Emerging ubiquitous services/applications will be provided based upon an open web-based service environment as well as legacy telecommunication and broadcasting services based. In particular, application programming interface (API) and web with dynamics and interactivities that do not exist today should be supported. Such a web-based service environment will allow not only creation of retail community-type services but also building of an open service platform environment which third-party application developers can access and launch their own applications. Using interactive, collaborative and customizable features, the web can provide rich user experiences and new business opportunities for the provision of ubiquitous networking services and applications.

- **Context-awareness and seamlessness capabilities**

Context-awareness implies the ability to detect changes in the status of objects. Intelligence system associated with this capability can help to provide the best service which meets the situation using user and environmental status recognition. Seamlessness is a capability that can be supported in many different ways: at the network level using handover and roaming in heterogeneous networks, at the device level with no service interruption during device changing and recognition, and at the content level for providing personalized content delivery services, e.g. based on user's situation, user's device, and network conditions.

- **Multi-networking capabilities**

Transport stratum needs multi-networking capabilities in order to simultaneously support unicast/multicast, multi-homing, and multi-path, etc. Because of high traffic volume and number of receivers, ubiquitous networking requires multicast transport capability for resource efficiency. Multi-homing enables the device to be always best connected using multiple network interfaces including different fixed/mobile access technologies. These capabilities can improve network reliability and guarantee continuous connectivity with desirable QoS through redundancy and fault tolerance.

- **End-to-end connectivity over interconnected networks**

For ubiquitous networking, it is critical to develop the solution to provide end-to-end connectivity to all of objects over interconnected heterogeneous networks such as fixed networks, broadcasting networks, mobile/wireless networks, etc. IPv6 with large address space can be considered as a good candidate for providing globally unique addresses to objects. IPv6 offers the advantages of localizing traffic with unique local addresses, while making some devices globally reachable by also assigning them globally scoped addresses.

5.3 Service Evolution Using Smart Environment of the Cloud and the IoT

Using smart environments of the cloud and the IoT, various services can be supported as shown in Figure 4. On top of our proposed cloud-based IoT, several kinds of smart services should be supported with the help of key functionalities such as service provisioning and management. We can represent it the cloud-based service of things which uses the cloud and the IoT.

There are two solutions for virtual resources of service provisioning and management in a cloud: horizontally with network virtualization and vertically with resource virtualization. For network virtualization [11], it is essential to develop the technology that enables the creation of logically isolated network partitions over shared physical network infrastructures so that multiple heterogeneous virtual networks can simultaneously coexist over the shared infrastructures. In addition, for resource virtualization, we also need to consider the virtualization of resources which include software, equipment, platform, computing, storage, memory, etc.

In conclusion, a novel resource management for service provisioning and management in a cloud will be a key enabler for realizing smart services of the IoT.

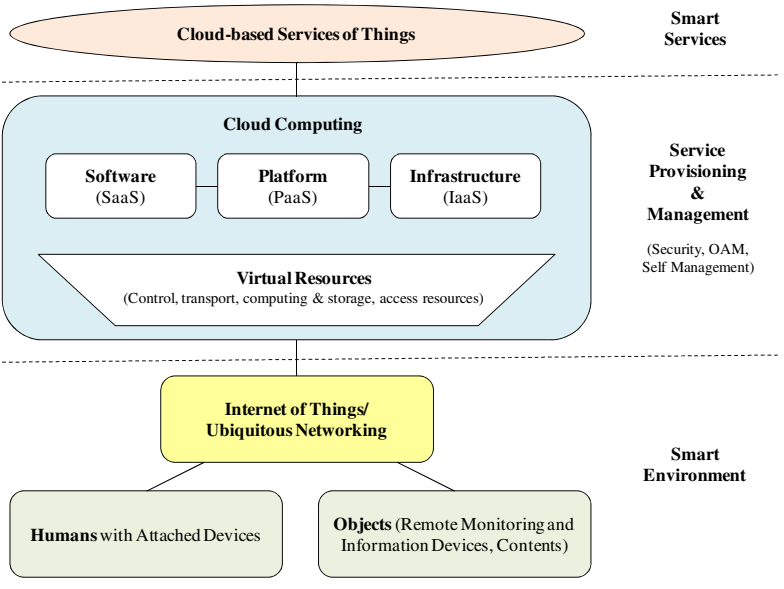


Fig. 4. A conceptual diagram for the cloud-based Internet of Things

6 Conclusion

This paper has presented characteristics of cloud computing in telecom perspectives and the issues to support ubiquitous networking for IoT. We have provided the basic concept and visions of ubiquitous networking and clearly identified key technologies essential to the ubiquitous networking in the IoT environment. For developing the relevant technical solutions, we have proposed the cloud-based IoT service environment which combines both the cloud computing and the IoT. We hope that our proposals will provide some key inputs for realization of IoT.

As future work, we plan to focus on objects-to-objects communications for various use cases using cloud computing in the IoT environment and business aspects. For this, it would be helpful if the relevant research efforts for realization of the cloud-based IoT are accelerated with special consideration of their commercial viability.

References

1. Lee, C.S., Won, R.: Future vision on "Beyond NGN" and next study period of SG13. ITU-T COM 13 – C 191 – E (April 2007)
2. Gubbins, How telcos could conquer the cloud (April 2009), `http://connectedplanetonline.com`
3. Knightson, K.: Question NEW-Q-ARCH – Principles and functional architecture for NGNs including ubiquitous networking. ITU-T TD 423 Rev1 (WP2/13) (January 2008)
4. ITU-T Internet Reports, Internet of Things (November 2005)

5. Cloud computing - wikipedia, http://en.wikipedia.org/wiki/Cloud_computing
6. Mell, P., Grance, T.: The NIST definition of cloud computing. version 15 (October 2009), http://csrc.nist.gov
7. Definition of Cloud Computing, incorporating NIST and G-Cloud views, http://www.katescomment.com/feed/
8. Mota, R.: Five telecom provider benefits of offering cloud computing services, http://searchtelecom.techtarget.com
9. Lee, G.M., Choi, J.K., Chung, T., Montgomery, D.: Standardization for ubiquitous networking in IPv6-based NGN. ITU-T Kaleidoscope Event - Innovations in NGN, pp. 351–357 (May 2008)
10. Lee, G.M., Choi, J.K., Jo, S.K., Kim, J.Y., Crespi, N.: Naming Architecture for Object to Object Communications. IETF internet-draft <draft-lee-object-naming-02.txt> (March 2010) (work in progress)
11. Network virtualization – wikipeida, http://en.wikipedia.org/wiki/Network_virtualization

Relay Placement Problem in Smart Grid Deployment

Wei-Lun Wang and Quincy Wu

Department of Computer Science and Information Engineering,
National Chi Nan University, Puli, Nantou, Taiwan

Abstract. In this paper, we give an overview of power grid, smart grid, Advanced Metering Infrastructure (AMI), and the deployment cost analysis step by step. The importance between Relay Placement Problem (RPP) and the deployment cost in an AMI system is highlighted. Additionally, a decision supporting system in a pilot AMI system funded by National Science Council (NSC) is briefly described where RPP is solved by an approximation algorithm.

1 Introduction to the Power Grid

The power grid is an interconnected network for delivering electricity from suppliers to consumers. Three processes are involved in the electricity delivery within the grid. That is,

- Electricity generation
- Electric power transmission
- Electricity distribution

As shown in Fig. 1, the initial step is to generate sufficient electricity in the generation station. After that, generated electricity is ready to be transmitted through transmission lines. The electricity of 765 kV, 500 kV, 345 kV, 230 kV, and 138 kV are sent out. In the final process, the electricity with high voltages is then distributed to sub–transmission customers with 26 kV and 69 kV, to primary customers with 13 kV and 4 kV, and to secondary customers like houses in residential areas with 120 V and 240 V.

For the traditional electrical grid mentioned above, there are some energy issues:

- Reliability
- Economy
- Efficiency
- Affordability
- Environmental Impacts

In the reliability aspect, blackout usually takes place in unexpected situations suddenly. The unstable power supplies could put heavy impact on the economy. According to the Electric Power Research Institute (EPRI) report in June 2001,

T. Margaria and B. Steffen (Eds.): ISoLA 2010, Part I, LNCS 6415, pp. 411–424, 2010.

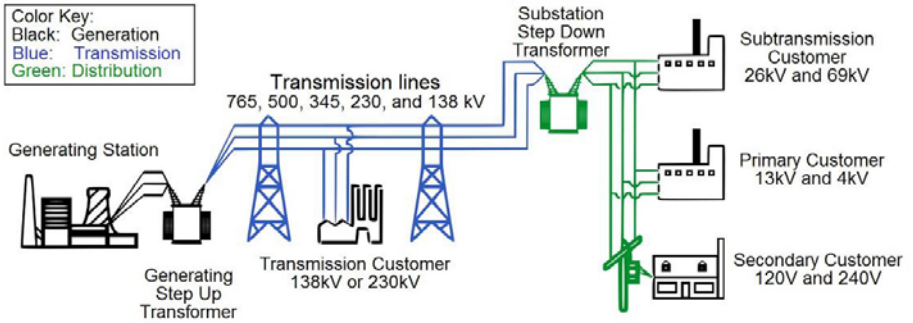


Fig. 1. Simple Diagram of Power Grid in North America [10]

"The Cost of Power Disturbances to Industrial & Digital Economy Companies" estimated that the annual direct cost of power outages and power quality disturbances for all sectors was between $120 billion and $188 billion. In its 2003 report, the Department of Energy (DOE) noted that "it is estimated that power outages and power quality disturbances cost the economy from $25 to $180 billion annually." [2]

There are some technical reasons which could be imputed to slow mechanical switches that cannot endure heavier load, no automated analysis for situational awareness, and so forth. Moreover, the electricity outage actually results from the overburdening since people need more and more electricity in life. However, it can be easily understood that electricity demand is not always the same during twenty–four hours in a day. According to the Pareto principle (also known as the 80–20 rule), about 80 percents of the total required electricity are consumed in 20 percents of time in one day. In other words, a large part of electricity is consumed in specific 4.8 hours. The power plant works hard merely in 0.2 day and relaxes in the remaining 0.8 day. Because of it, the efficiency of the power grid is extreme low.

Even worse, the cost of electricity goes up as the price of fossil fuel rises. When people work hard to earn money, factories increase their output. At the same time, more electricity is needed. Currently most electricity is generated by burning coals, which will emit a great deal of greenhouse gas (GHG) that traps heat in the atmosphere. It can be seen that the temperature on Earth's surface will go up and up because of producing too much GHG.

Therefore, to improve energy utilization efficiency and further improve our lives, a better power grid is desired. That is why the smart grid was proposed for efficiently utilizing the limited resources in generating electricity.

2 Smart Grid

Like the power grid, the smart grid also delivers electricity from suppliers to consumers. However, it uses two–way digital technology to support communication between consumers and suppliers. It combines power systems, telecommunications, smart energy devices, information technology, and digital control (Fig. 2).

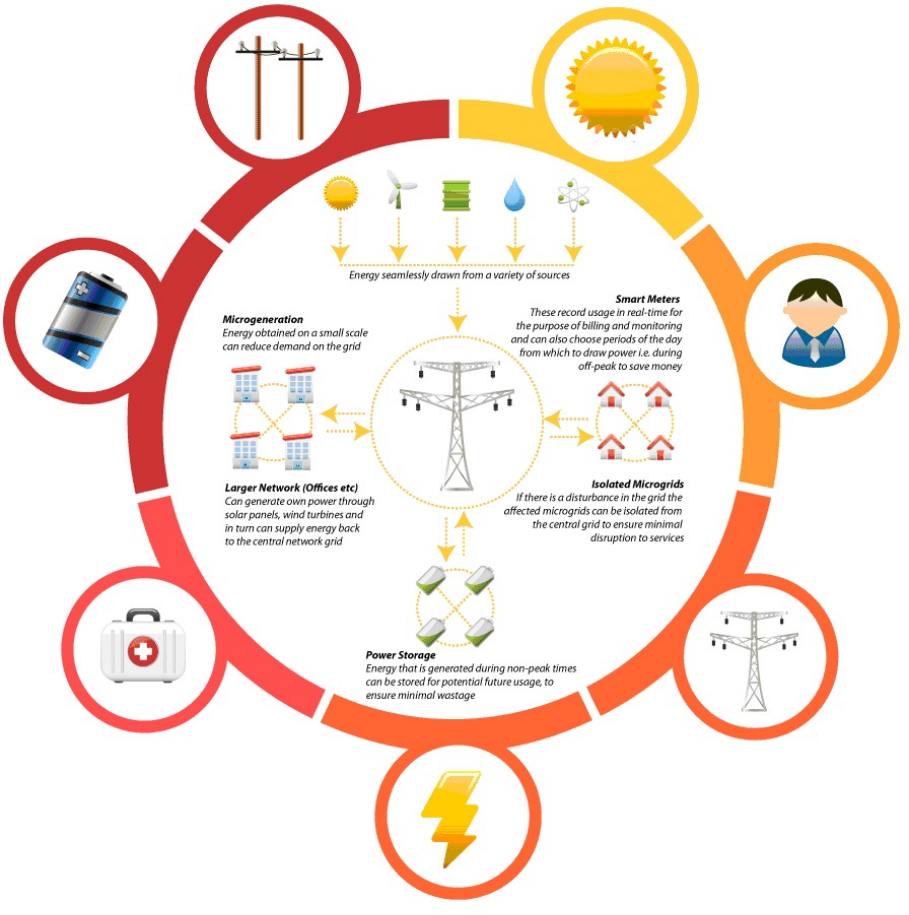


Fig. 2. Smart Grid [11]

There are some characteristics about the smart grid [3]:

- Sensing and Measurement
- Integrated Communication
- Improved Interfaces and Decisions Support
- Advanced Components
- Advanced Control Methods

First of all, through intelligent devices like electricity meters, all suspicious behaviors can be detected in real time via sensing and measurement, such as consuming large amount of electricity at night when no electrical equipment is expected to be powered on. Secondly, it contains an integrated two–way communication network which allows consumers to monitor all electricity flowing in the system and even control smart components such as intelligent washing machines. Intelligent machines like these can run at scheduled time. Since the policy of Critical

Peak Pricing (CPP), where higher price for electricity is charged in peak hours in contrast with off-peak periods, is used for charging electricity, and intelligent machines can be turned off in peak hours and turned on while the electricity price is least expensive, this system can surely bring economical benefits to consumers. Similarly, suppliers can also communicate with consumers to check whether people agree to turn off some appliances in peak hours; people who agree to do so will be rewarded with some "credit points." Besides that, there are many advanced components in a smart grid, such as superconductivity transmission lines and storage facilities. Suppose each consumer has a large–size battery in his/her house, which can store electricity during off–peak time. Then during peak time, consumers can get the electricity directly from the battery, or even sell it back to suppliers. This would significantly reduce the demand of electricity from suppliers during the peak hours. Finally, advanced control methods would be designed to handle the whole smart grid system through integrated communications.

Foreseeing the importance of this new technology, many countries including Italy, Sweden, Holland, and England are developing nation–wide smart grid projects. Some other countries are carrying out smaller–scale regional smart grid plans. In America, President Obama put $3.4 billion into the research and deployment of smart grids in October, 2009. Fig. 3 shows the location of these

Fig. 3. AMI/AMR Deployment around U.S.A. [12]

pilot projects of automatic meter reading (AMR) and advanced metering infrastructure (AMI).

3 Advanced Metering Infrastructure (AMI)

To build a smart grid system, an advanced metering infrastructure (AMI) is required to actively measure, collect, and analyze energy consumption data. AMI has the interface to interact with advanced devices such as electricity meters, heat meters, gas meters, water meters, and any other intelligent appliances. In this section, we shall introduce how AMI works and the architecture of its communication networks. A pilot AMI project funded by National Science Council (NSC) in Taiwan will be demonstrated as a real example.

3.1 How AMI Works

Generally, an AMI system consists of four components (Fig. 4) – the smart meter, the energy display and controller, the communication network, and the meter data management application (MDMA).

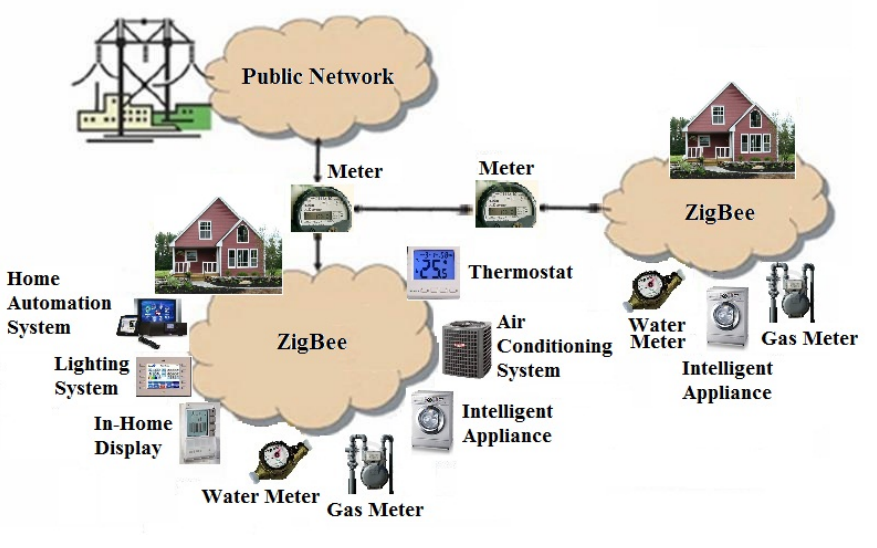


Fig. 4. An Illustration of AMI

The smart meter is an electricity meter with additional capacity to record energy usage and receive messages from suppliers. Besides, it can be turned on and off via remote controllers. Data cumulated by smart meters would be shown on the smart meter directly (Fig. 5) or on controllers such as the in–home display (IHD) device (Fig. 6). Consumers could interact with suppliers or check energy usage data through this device. The success of an AMI system would rely on a reliable communication network to connect smart meters with the display and controllers, so that the MDMA can manage collected data to make wise decisions.

Fig. 5. Smart Meter [13]

Fig. 6. In-Home Display [14]

3.2 National Science Council Program in Taiwan

To foster the AMI technology in Taiwan, National Science Council (NSC) funded some universities to run a pilot project inside the campus. The NSC project consists of four subprojects:

- Smart Power–saving Outlets
- Small–scale Energy Storage System
- Network and Communication Technology
- Smart Energy Management Application Service Platform

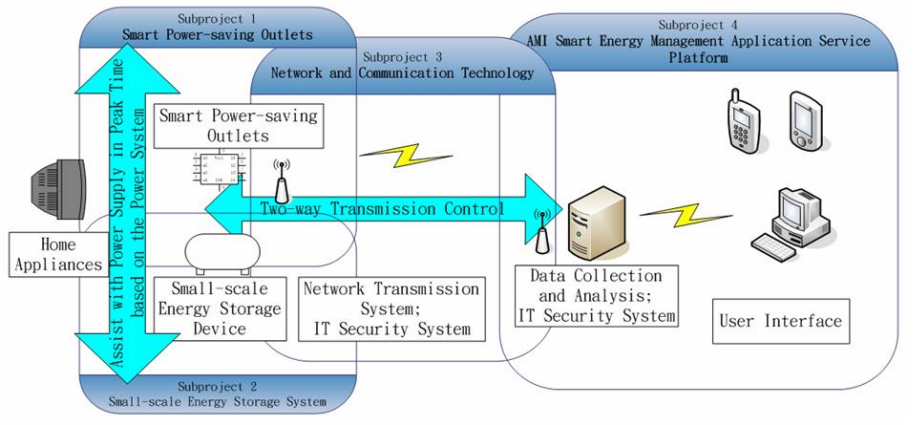


Fig. 7. Subprojects in the NSC AMI project in Taiwan

Fig. 7 shows the relationship between four subprojects of an AMI system. In the 1^{st} subproject, smart outlets will be designed which can intelligently record the electricity usage and then transmit the data to the management application of the 4^{th} subproject, where electricity usage data are stored and analyzed. According to the received data, the management application can monitor and control the overall electricity usage and even detect abnormal situations such as the excessive consumption of electricity on the outlet. Besides, to efficiently utilize the power and balance the electric output of power suppliers, a small–scale energy storage system for consumers is developed in the 2^{nd} subproject. This allows consumers to charge electricity in the off–peak time. During the peak time, consumers can get electricity directly from the storage system (usually some batteries) without demanding electricity from suppliers, and this can reduce the load of power suppliers in the peak time. As data transmission, the 3^{rd} subproject is responsible for making a two–way communication between management application and smart devices, smart outlets, and energy storages. Through the technology of the two–way communication, consumers can acquire data from smart devices, and put commands to control them, too.

3.3 Factors That Affect Deployment Cost

No matter how intelligent and wonderful the smart grid is, the cost of deployment and maintenance will be a key factor which will determine whether this new technology will be successful and widely adopted. As shown in Fig. 8, facility hardware and network hardware are the major cost of an AMI deployment, which are 45% and 20%, respectively.

Many AMI communication networks adopt the ZigBee wireless technology because of its low power and low cost characteristics. However, as every wireless technology, ZigBee suffers from the signal decay problem. Therefore, we need a relay which is the device used to strengthen the received signal and then re–transmit it

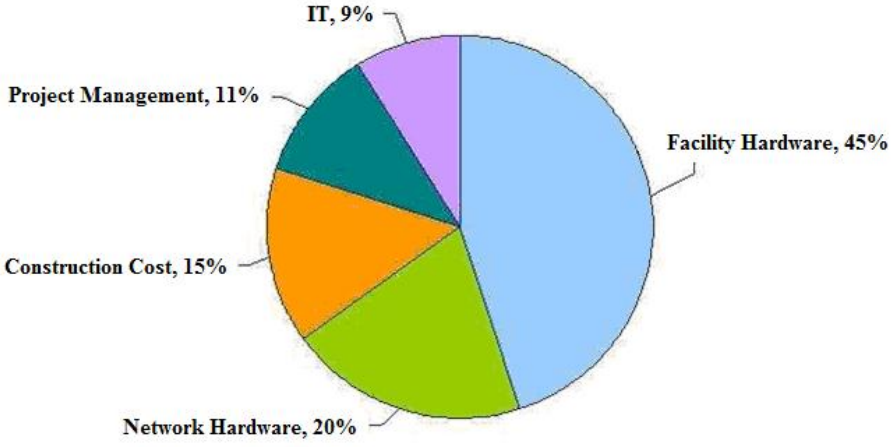


Fig. 8. AMI Deployment Cost Analyzing Illustration [15]

to the destination. Because of the limited range for radio communication, usually many relays must be added to form a connected stable network. Suppose the main facilities like smart devices, storages, and management applications are fixed, one factor that can affect the cost of the communication network is the number of relays. If the relays are not wisely deployed, quite a large number of relays must be deployed to form a connected network, which will significantly increase the cost of the AMI communication network. In next section, we shall formally define the Relay Placement Problem and illustrate the algorithms to solve it.

4 Relay Placement Problem (RPP)

The problem of finding positions of minimum number of relays in a smart grid system is called RPP. The problem definition, evaluation criterion of approximation algorithms, variations of RPP, and related algorithms are explained below.

4.1 Problem Definition

The definition of RPP is shown as follows: On a Euclidean plane, given a set of N sensors, which have the effective communication range 1, and a fixed number $R \geq 1$, which is the effective communication range of a relay. RPP is to place a minimum number of relays so that between every pair of sensors there is at least a path through relays such that the consecutive vertices of the path are within distance R if both vertices are relays and within distance 1 otherwise, forming a connected network. Fig. 9 shows a connected network with smart devices and relays.

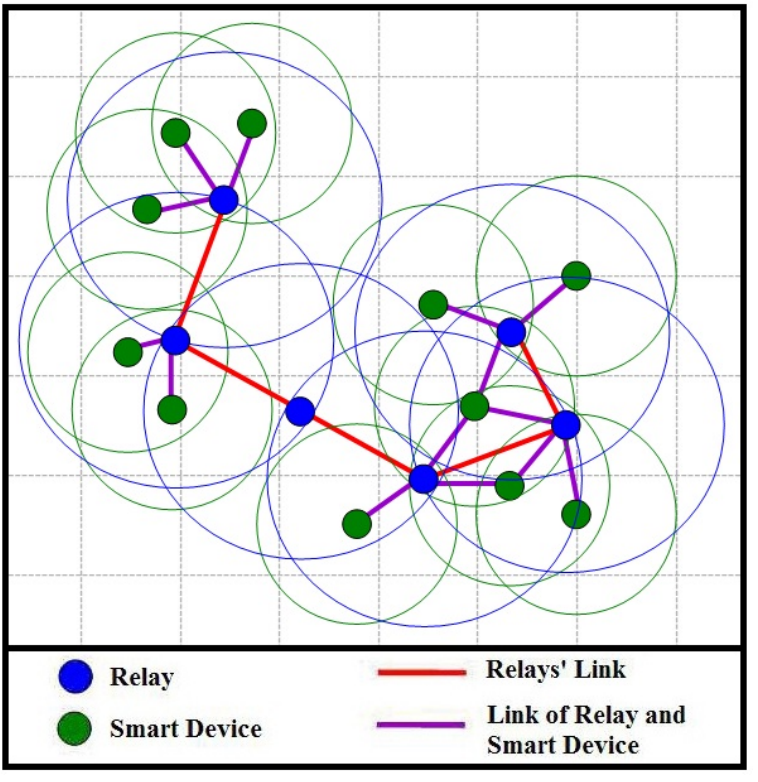


Fig. 9. Relay Placement Problem

4.2 The Worst Situation of Deployment

This subsection demonstrates some bad situations when relays are not deployed wisely. Assume that there are N sensors, and the effective communication rage of a relay is R. Those $N-1$ sensors form a standard circle where the distances of consecutive sensors are D, where $1 < D \leq 2$. The remaining sensor is placed in the center of the circle. Therefore, the distance between the central sensor and each outside sensor is approximately $\frac{(N-1)D}{2\pi}$.

To make the whole N sensors connected, relays can be placed between the central sensor and each of the $N-1$ outside sensors. Since the distances between central sensor and other sensors are all $\frac{(N-1)D}{2\pi}$, $(N-1)\lceil\frac{(N-1)D}{2\pi R}+1\rceil$ relays are needed to form a connected communication network. (Fig.10)

However, there is another approach to deploy relays in this case. Suppose we place a relay in the middle of consecutive outside sensors. Totally $N-1$ relays will be deployed at this step. After that, we connect the central sensor to one another sensor arbitrarily via $\lceil\frac{(N-1)D}{2\pi R}\rceil+1$ relays. Through this way, all sensors become connected by only $N+\lceil\frac{(N-1)D}{2\pi R}\rceil$ relays, as shown in Fig.11.

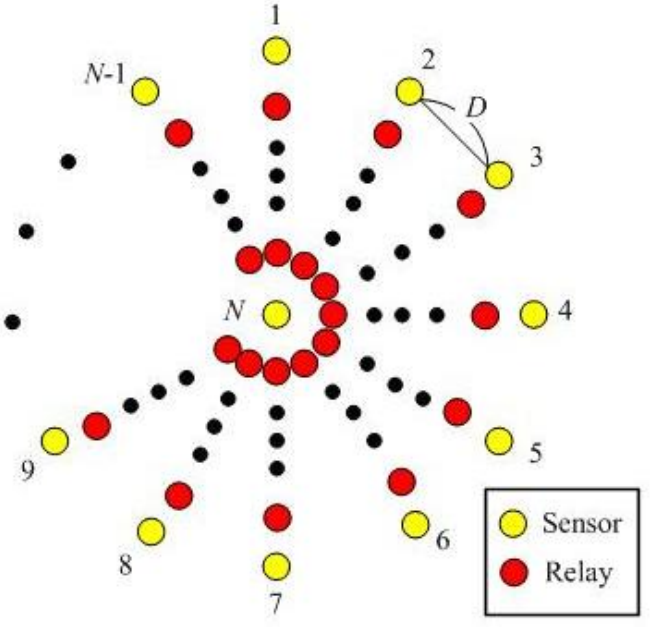


Fig. 10. A Bad Deployment of Relays

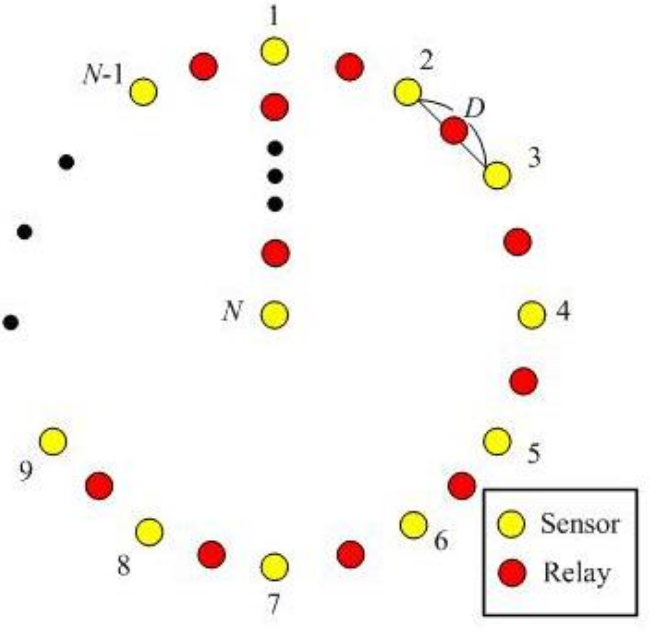


Fig. 11. A Wise Deployment of Relays

The above two deployment approaches are extremely different. They will require $(N-1)\lceil\frac{(N-1)D}{2\pi R}+1\rceil$ and $N+\lceil\frac{(N-1)D}{2\pi R}\rceil$ relays, respectively. Assume $N=1000$, $D=2$, and $R=4$. The required number of relays for the first approach is approximately 80919; however, the second approach only needs 1080 relays. That is to say, people who adopt the first approach to deploy relays must spend nearly 75 times higher cost than those who choose the second approach.

4.3 The Evaluation Criterion – Approximation Ratio

Since Minimum Geometric Disk Cover (MGDC) problem is a special case of RPP, and MGDC is known to be NP–complete [9], RPP is an NP–hard problem. That is, it is unlikely for us to find a polynomial–time algorithm which returns the optimal solution. Therefore, many approximation algorithms for solving RPP were proposed.

It is important to define an evaluation criterion to judge which approximation algorithms perform better. If an approximation algorithm A guarantees to return a solution bounded by α multiplies by the optimal solution, then A is said to be an α–approximation algorithm and has an approximation ratio of α. Therefore, the approximation algorithm which has the smaller approximation ratio is desired here since RPP is expected to deploy fewer relays in a communication network.

4.4 Approximation Algorithms and Variations of RPP

For the simplest RPP problem, forming a connected network is enough. However, considering higher reliability in the communication network of a smart grid, one variation of RPP, namely 2–connected RPP, which is to form a 2–connected network where there exist two disjoint paths between any two sensors, was also proposed.

Table 1. Approximation Ratio Comparisons

	LWJ2006 [4]	THS2006 [5]	LX2007 [6]	EFGMPS 2008 [7]	CDWX 2008 [8]
RPP	$6+\epsilon$	8 and 4.5	7	3.11	3 and 2.5
2–Connected RPP	$(24+\epsilon)$ and $(\frac{6}{T}+12+\epsilon)$	6 and 4.5			

To the original RPP, [4,5,6,7,8] proposed $(6+\epsilon)$–approximation, 8–approximation, 4.5–approximation, 7–approximation, 3.11–approximation, 3–approximation, and 2.5–approximation algorithms respectively, as shown in Table 1. Fig. 12 depicts an example that runs the 8–approximation algorithm in [5] to deploy relays.

To the 2–connected RPP, [4] proposed a $(24+\epsilon)$–approximation and a $(\frac{6}{T}+12+\epsilon)$–approximation algorithm, in which T is the ratio of the number of relays placed in its RPP to the number of sensors, while [5] proposed a 6–approximation and a 4.5–approximation algorithm.

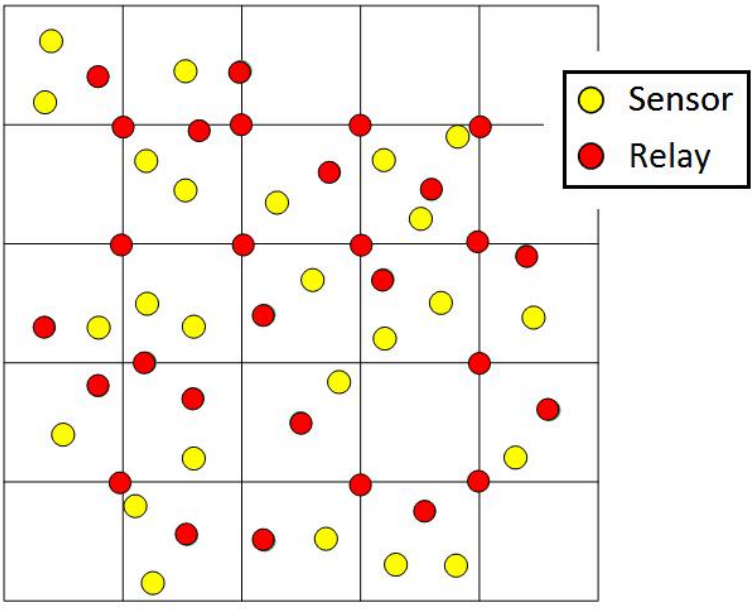


Fig. 12. An Example of [5]'s 8–approxiamtion Algorithm

4.5 Decision Supporting System in NSC Program

In our NSC AMI project, a 4.5–approximation algorithm of 2–connected RPP in [5] is selected to serve the decision supporting system for relay deployment because of the easy practice and reliability. The steps are described briefly below.

Algorithm 1. A 4.5–Approximation Algorithm for RPP

Step 0. Divide the region into cells with side length 4. For each cell, find all P–positions for relays (where position Y is called a P–position if there exist two sensors which are one unit away from Y). Move those P–positions outside the cell to the nearest border on its cell.

Step 1. Inside each cell, exhaustively search all 1 through 21–subsets of the P–positions inside (or on) the cell, to find a subset with smallest order which can cover all sensors in the cell at least twice. For each cell B_i, Let H_i be the set of relays found for B_i (note that each H_i has at least two relays).

Step 2. Inside each cell B_i, if the relays in H_i are connected but not 2–connected, add a relay on the horizontal mid–line of B_i, 4 units away from the left–top corner of B_i. If they are not connected, then add one more relay on the vertical mid–line of B_i, 4 units away from the left–top corner of B_i.

Step 3. Now, we make the set of the chosen relays to be connected.

Step 3.1. Connect each row of the H_i's.

Let H_{i+1} be the set of relays to the right of H_i. If H_i and H_{i+1} are not connected, add a relay, q_i, on the horizontal mid–line of B_i, 4 units away from the left–top corner of B_i, and set $H_i = H_i \cup q_i$. If they are still not connected, add a relay, q_{i+1}, on the horizontal mid–line of B_{i+1}, 4 units away from the left–top corner of B_{i+1}, and set $H_{i+1} = H_{i+1} \cup q_{i+1}$ (notice that we only make them connected here, instead of 2–connected).

Repeat **Step 3.1**, until every row of H_i's is connected.

Step 3.2. Connect each column of the H_i's.

For each H_i, let $LEFT_{B_i}$ denote the set of relays in H_i, which are connected to some relays in the set to the left of H_i, i.e., $LEFT_{B_i} = \{q \in H_i | q$ is connected to a node in the set to the left of $H_i\}$; let $RIGHT_{B_i}$ denote the set of relays in H_i, which are connected to some relays in the set to the right of H_i, i.e., $RIGHT_{B_i} = \{q \in H_i | q$ is connected to a node in the set to the left of $H_i\}$.

If $|LEFT_{B_i} \cup RIGHT_{B_i}| > 1$, then H_i'=H_i. Otherwise, $H_i' = H_i - LEFT_{B_i} \cup RIGHT_{B_i}$.

Let H_{i+x} be the set of relays directly under H_i. If H_i' and H_{i+x}' are not connected, then add a relay, q, on the vertical mid–line of B_i, 4 units away from the left–top corner of B_i, and set $H_i = H_i \cup \{q\}$. If they are still not connected, add a relay, q', on the vertical mid–line of B_{i+x}, 4 units away from the left–top corner of B_{i+x}, and set $H_{i+x} = H_{i+x} \cup \{q'\}$.

Repeat **Step 3.2**, until every column of H_i's is connected.

Suppose the optimal deployment of a RPP problem requires N relays. Mathematically, it can be shown that the above algorithm can always give a deployment with at most $4.5N$ relays. [5]

5 Conclusions and Future Works

The deployment cost of a smart grid system is crucial to its success. This paper illustrated the Relay Placement Problem, which significantly affects the cost of an AMI system when ZigBee is adopted in its communication network. To solve the Relay Placement Problem, we demonstrated a decision supporting system in our NSC project, which adopted intelligent algorithms to minimize the number of relays in question. When deploying larger networks, where significant amount of relays are needed, an efficient RPP algorithm would be very important to control the deployment cost.

Acknowledgments

This work was supported in part by National Science Council in Taiwan under grants NSC98-2218-E-029-004 and NSC99-2221-E-260-012.

References

1. Frye, W.: Smart Grid – Transforming the Electricity System to Meet Future Demand and Reduce Greenhouse Gas Emissions. Cisco Internet Business Solutions Group (November 2008)
2. Reitenbach, G.: Smart Grid – On the Money (April 1, 2010), http://www.powermag.com/smart_grid/Smart-Grid-On-the-Money_2578.html
3. Pullins, S.: Key Technologies for a Modern Grid (October 10 (2006), http://www.smartgridnews.com/artman/publish/article_172.html
4. Liu, H., Wan, P., Jia, X.: On Optimal Placement of Relay Nodes for Reliable Connectivity in Wireless Sensor Networks. Journal of Combinatorial Optimization 11, 249–260 (2006)
5. Tang, J., Hao, B., Sen, A.: Relay Node Placement in Large Scale Wireless Sensor Networks. Computer Communications 29, 490–501 (2006)
6. Lloyd, E., Xue, G.: Relay Node Placement in Wireless Sensor Networks. IEEE Transactions on Computers 56, 134–138 (2007)
7. Efrat, A., Fekete, S., Gaddehosur, P., Mitchell, J., Polishchuk, V., Suomela, J.: Improved Approximation Algorithms for Relay Placement. In: Halperin, D., Mehlhorn, K. (eds.) Esa 2008. LNCS, vol. 5193, pp. 356–367. Springer, Heidelberg (2008)
8. Chen, X., Du, D., Wang, L., Xu, B.: Relay Sensor Placement in Wireless Sensor-Network. Wireless Networks 14, 347–355 (2008)
9. Fowler, R.J., Paterson, M.S., Tanimoto, S.L.: Optimal packing and covering in theplane are NP-complete. Information Processing Letter 12, 133–137 (1981)
10. http://www.inlanddevelopment.com/electricfacts.html
11. http://www.ngpowereu.com/news/smart-grid-revolution/
12. http://www.treehugger.com/files/2008/11/smart-grids-who-is-onboard.php
13. http://www.reuk.co.uk/Smart-Electricity-Meters.htm
14. http://www.centrica.co.uk/index.asp?pageid=76
15. http://www.topology.com.tw/tri/ (Topology Research Institute)

Towards a Research Agenda for Enterprise Crowdsourcing

Maja Vukovic[1] and Claudio Bartolini[2]

[1] IBM T.J. Watson Research,19 Skyline Dr, Hawthorne, NY, 10532, USA
maja@us.ibm.com
[2] HP Labs, Palo Alto, CA, USA
claudio.bartolini@hp.com

Abstract. Over the past few years the crowdsourcing paradigm has evolved from its humble beginnings as isolated purpose-built initiatives, such as Wikipedia and Elance and Mechanical Turk to a growth industry employing over 2 million knowledge workers, contributing over half a billion dollars to the digital economy. Web 2.0 provides the technological foundations upon which the crowdsourcing paradigm evolves and operates, enabling networked experts to work collaboratively to complete a specific task. Enterprise crowdsourcing poses interesting challenges for both academic and industrial research along the social, legal, and technological dimensions.

In this paper we describe the challenges that researchers and practitioners face when thinking about various aspects of enterprise crowdsourcing. First, to establish technological foundations, what are the interaction models and protocols between the Enterprise and the crowd. Secondly, how is crowdsourcing going to face the challenges in quality assurance, enabling Enterprises to optimally leverage the scalable workforce. Thirdly, what are the novel (Web) applications enabled by Enterprise crowdsourcing.

Keywords: crowdsourcing, business process modeling.

1 Introduction

Crowdsourcing, a successful mechanism to harvest information and expertise from the masses in domains ranging from gold mining to t-shirt design, is becoming increasingly popular, and is attracting global enterprises given a promising, low cost, access to scalable workforce online. Crowdsourcing takes many different shapes and forms, from mass data collection to enabling end-user driven customer support.

In order to understand and map out research challenges, we first describe the crowdsourcing landscape and use it distill a set of technical, legal and social challenges that need to be overcome for enterprises to successfully adopt crowdsourcing. This paper builds on the current advances in crowdsourcing research and incorporates discussions from the First International Workshop on Enterprise Crowdsourcing, held in conjunction with Tenth International Conference on Web Engineering, in Vienna in July 2010 [1].

T. Margaria and B. Steffen (Eds.): ISoLA 2010, Part I, LNCS 6415, pp. 425–434, 2010.

The remainder of this paper is structured as follows. Section 2 provides an overview of different applications of enterprise crowdsourcing, while Section 3 discussed the variants of crowdsourcing process, such as competition and result aggregation. Section 4 identifies three broad classifications of the crowd, which necessarily require and drive the design of the suitable crowdsourcing approach (e.g. incentives and governance processes). Section 5 discusses incentives schema and their affect on the success of crowdsourcing effort. Section 6 presents the landscape of challenges in quality assurance when engaging masses online for data collection or problem solving. Section 7 lays out a set of open questions with respect to (corporate and IT) governance, as the crowdsourcing is being adopted by enterprises. Section 8 discusses existing research in social aspects of crowds, such as demographical diversity. Section 9 summarizes state of the art in known business models that are supporting enterprise crowdsourcing. Section 10 reviews the open questions posed in this paper.

2 Applications of Crowdsourcing in the Enterprise

In this section, we briefly review some of the literature on successful and interesting applications of crowdsourcing projects for enterprises. This helps us set the stage for the remainder of the paper, where we will use these examples to illustrate some of the concepts we introduce. For a more comprehensive account, Vukovic [2] offers a thorough categorization of existing crowdsourcing examples from enterprise domain, based on the type of activity that is being crowdsourced and the relationship to stages in product/service lifecycle.

Karnin et al. [3] present an application of crowdsourcing to the processing of scanned documents within the support services function of an enterprise.

Lopez et al. [4] illustrate two use cases of enterprise crowdsourcing, one that employs wisdom of crowd approach for knowledge intensive processes, and another that proposes marketplaces for customer support services. Lopez et al., propose a *PeopleCloud* framework to support crowdsourcing within enterprises, and identify number of roles and processes that need to be in the place.

Maintaining an up-to-date IT Inventory is a challenging task, as the key experts transition into new roles within or outside the enterprise. Vukovic et al. [5] deployed an application of crowdsourcing to the IT Asset Management problem. Using the wisdom of crowd approach they effectively engaged more than 2500 enterprise experts to map out business capabilities of a physical IT infrastructure, which is hosting thousands of applications on hundreds of thousands of servers. This has introduced a performance improvement of 30x in comparison to the traditional, manual approach of reaching out to experts. More importantly, beyond the cost savings arising from the process automation, the value of data collected is significant, as a driver for other IT optimization processes. In addition, the crowdsourcing process has captured social networks in the enterprise, and generated communities of experts centered on the IT asset repository.

Stewart et al. [6] demonstrate another powerful application for enterprises – they employ participants within a large global enterprise to collect translation data sets, in order to improve the translation algorithms. Based on their experience in running

internal enterprise challenges, and observations of participant behavior they propose an incentive model to cater for a diverse crowd.

GiffGaff and CrowdEngineering [7] are examples of how end-user crowd can be engaged in customer support scenarios.

Brightidea.com provides a platform for crowd members to share their ideas and compete with other for material prizes, and eventually turn their ideas into marketable products. In contrast, InnoCentive.com provides a virtual place for crowd to develop ideas, designs, prototypes or final products. It attracts research and business communities.

Prediction markets, such as BRAIN [8], Crowdcast.com, Marketocracy.com, Intrade.com and Pickspal, employ information aggregation technology to harnesses the power and truth-telling properties of market mechanisms.

3 Models of Crowdsourcing

A generic exercise in crowdsourcing consists in breaking down a problem or a goal into lower level, smaller sub-tasks to be executed by a crowd. The outcomes of each of the sub-tasks have to be aggregated (reconciled, re-composed) in order to provide an answer to the original problem or satisfy the original goal. Therefore, one first fundamental dimension to categorize different models of crowdsourcing is the nature of the step of aggregating intermediate outcomes returned by the crowd members into the final outcome of the crowdsourcing exercise.

Broadly speaking, there are two mutually exclusive ways of aggregating partial outcomes into a final outcome. In one, partial outcomes **are composed** into the final outcome. That means that the eventual project outcome is a functional composition of input got from the results of the sub-tasks sourced to the crowd. In this case, most, if not all, or the results returned from the crowd are necessary to derive the solution sought. In the other extreme, partial outcomes **compete** to be selected as the final solution (or parts thereof). In actuality, often crowdsourcing projects use elements of both aggregation models, resulting in a **hybrid** approach.

Wisdom of crowds

The model of crowdsourcing where the wisdom of crowds [9] is exploited to its full power is the model where outcomes are composed together. The emerging property of this crowdsourcing model is that the crowd is able to find an answer to the original question asked that none of its members could have (that is the crowd is "smarter" than all of its members. Applications of this model include prediction markets (BRAIN [8], and other listed in the previous section), along with Crowdsourcing projects where sub-tasks outcomes are aggregated in more straightforward ways (e.g. GasBuddy.com, WNYC's Cheapest Gallon of Milk).

Contest/Marketplace

When member's outcomes compete to determine the final crowdsourcing project outcome, we have models such as contests, or marketplaces. The most comprehensive model to date of crowdsourcing context is given in Archak et al. [10]. Carpenter [11]

also analyzes crowdsourcing contests, and categorizes them based on the kind of decisions that result in selecting the best outcomes: a) *crowd sentiment*, *expert decision*, b) *crowd decision*, c) *expert decision* or d) *American idol*.

In marketplace crowdsourcing, such as Amazon.com Mechanical Turk, the aggregation of results follow the market mechanism set by the marketplace, therefore a study of the diverse possibilities for aggregation (selection) of partial results can be undertaken from the economic theoretical disciplines of mechanism analysis and mechanism design.

Hybrid aggregation models are often used in more complex crowdsourcing projects as in the case of business process crowdsourcing exemplified by CrowdEngineering ([7]).

There are interesting implications on how to set incentives for wisdom-of-crowds crowdsourcing projects versus contest/marketplace projects which we will discuss in section 4 below. Here it will suffice to note that in context/marketplace projects, incentives are usually set for participants to do carry out their sub-task in the best possible way, on the promise that they will get a reward only if their product is chosen among others. On the other hands, in wisdom-of-crowds projects, participants need to be incented to carry out their task to the best of their ability, even though the result that they carry out have no direct bearing on whether they will get a reward or not. E.g. the incentive for GasBuddy.com participants has to be roughly independent on whether one particular member happens to stumble upon the cheapest gallon of gas in town, whereas in logo selection contexts (e.g. Threadless.com), the reward given to the participant is function of the success of their outcome.

Aggregation mechanisms have also a bearing on quality control, as it is exemplified by Kern's work [12], discussed further in section 6.

4 Crowd Types

Another important dimension along which we seek to categorize enterprise crowdsourcing effort is the affiliation of the crowd with respect to the crowdsourcing enterprise. The easiest way to categorize crowd members is based on what kind of contract they have with the enterprise. Here we define an **internal** crowd as exclusively composed of employees of the crowdsourcing firm. For the sake of this definition, we will not make a distinction between actual employees of the crowdsourcing firm and contractors or other kind of part time employees. An **external** crowd is obviously defined as exclusively composed of members who are not employees of the crowdsourcing enterprise. To complete our model, we add the concept of a **hybrid** crowd, which is composed of both employees and non-employees, and as we will see in the following presents some characteristics of either extreme model.

The different types of crowd have implications on the following aspects:

- Incentives
- Intellectual property rights
- Information security

As we look at these aspects one by one, we highlight the differences between relying on an internal crowd versus and external one. The way in which the two types

combine into a hybrid crowd will depend on the actual crowdsourcing model used (see section 2 and 3).

Incentives

Incentives are usually easier to set for an external crowd. Reason for it is that there is the time and effort that external crowd members put into the crowdsourcing exercise has to be rewarded somehow. Therefore, the discussion about incentives that we expand in the next section, applies generically to external crowd. The situation is made more complicated for internal crowds. Reason for it being that, because the crowdsourcing enterprise is already paying for their employees' time and services, they might think that they may get away with not offering extra incentives for taking part in the exercise. For this, most of the non-monetary incentives that we describe in the next section do apply.

Intellectual property rights

For an internal crowd, intellectual property rights are usually regulated by the employment contract. For an external crowd, management of intellectual property right is a concern that is better taken care of upfront: see for example the way that Threadless.com deals with intellectual property rights attribution. Threadless.com – the crowdsourcing enterprise - must also protect itself from the possibility that works from crowd members that they decide to produce might infringe copyrights owned by third parties.

Information security

For an internal crowd, usual security policies might apply. For example, in the prediction market example of having a group of employees from various part of the business bid on revenue levels for the next quarter, the sensitivity of the information is such that the enterprise will certainly have policies that need to be followed in terms of sharing sensitive information. The situation is more interesting for external crowds. Tradeoffs have to be made between sharing information that is potentially sensitive to the crowdsourcing enterprise with external members, and putting the external members themselves in condition to do the best possible job.

5 Incentives

Incentives are a particularly important aspect of crowdsourcing projects, because they can make all the difference between success and failure. The principal dimension along which types of incentives can be split is **monetary** vs. **non-monetary**.

For some of the models of crowdsourcing, setting monetary incentives simply amounts to determining the right price to pay for one's services. In the Amazon.com Mechanical Turk example, since we are talking about a marketplace for micro tasks, then the monetary incentive is set as the value of the micro task to the crowdsourcing enterprise. The situation becomes more interesting when we start considering crowdsourcing contests. Now, the price to be paid to the winner of the context can be set in

advance – therefore based on the value of the project outcome to the crowdsourcing enterprise, or determined based on other factors, such as the perceived quality of the results (see discussion on quality assurance, in the next section). Again, for other models of crowdsourcing such as wisdom-of-the-crowd type of models, the monetary incentive to be paid has direct bearing to the quality of the outcome. An example of this is a prediction market crowdsourcing project of the kind that BRAIN [8] or Crowdcast.com's Social BI enable, which work on a *thinner* version but under the same principles of prediction markets such as the Iowa Electronic Markets. Because in prediction markets, the correct functioning of the market is a necessary pre-condition to guarantee the quality of the outcome, incentive for participation and for correct predictions have to be set correctly.

Even though they are not money, oft-used variants of brownie points can be assimilated to money, especially when accumulating them results in qualifying for prizes.

Non-monetary kinds of incentives include: establishing reputation (e.g. in particular academic reputation); alignment of one member's personal objectives with the objectives of the crowdsourcing project; non-profit driven motivation such as goodwill or personal fun., opportunity to socialize and be member of a community.

Goodwill and establishing reputation are the engines behind efforts such as Wikipedia and open source software development. Making crowdsourcing projects into games is also an effective way of guaranteeing participation. See for example [13].

Personal benefit in the exercise is (indirectly) part of the motivation in Wikipedia-like systems, but more relevantly this is the case in crowdsourcing projects such as the famous "Cheapest gallon of milk in town" exercise launched by Manhattan's WNYC radio station, or the Gas Price Feeds of GasBuddy.com. The motivation in such exercises is similar to that arising from volunteering information to researchers filling in questionnaires in exchanges for sharing into the results under the promise of being shared the aggregated results with.

6 Quality Assurance

The pervasive challenge in crowdsourcing is the ability to verify the data correctness and ensure the quality of contributions. This is a general problem and it spans different crowd types and their magnitudes. The quality of contributions is a key factor when evaluating the value proposition of using crowdsourcing, in particular for business objectives that are on the critical path, and the approaches range from automated selection algorithms to engaging the crowd for reviewing process.

Quality control approaches vary depending on the type of the crowdsourcing task. Majority of existing research focuses on the quality assurance for micro-tasks, in the domain of massive data collection, such as natural language annotation, image labeling and data labeling. Common mechanisms to evaluate results of these tasks include majority voting and aggregation of contributions [14].

Snow et al. [15], applied crowdsourcing to collect annotations for natural language processing, and compare the contributions of the crowds (such as ten participant contributions) to the expert labelers.

Sheng et al. [16], evaluate repeated acquisition of labels for data points when labeling is imperfect. Their results show that multiple label acquisition can be proven to be a beneficial strategy for label-quality and cost-regime trade-offs.

Kittur and Kraut [17] focus on human based approach to quality of contributions to Wikipedia. They examine how number of editors and the coordination methods they use influences the article quality, and highlight its significance to effectiveness in applying crowdsourcing.

Kern et al [12], analyze existing methodologies and propose a novel, matrix-based model for classifying quality assurance approaches to enterprise crowdsourcing, and identifying the corresponding mechanism for engaging the crowd for data assurance. They use four decision criteria to select the quality assurance approach, namely: determinacy of the task result, evaluation of execution and validation effort, required level of quality and number of equivalent relevant entries.

Chen et al. [18] address an interesting problem of data quality in crowdsourcing, when applied to the task of user studies, such as users' evaluating multimedia systems. Such studies are often impractical and costly. However, while crowdsourcing promises to be a low-cost solution, the probability of erroneous feedback (esp. when traded for monetary prizes) increases, given the lack of supervision. Chen et al propose a transitivity satisfaction rate (TSR) as a quantification of a participant's consistency throughout a run of an experiment.

More complex tasks, such as software development on TopCoder.com, rely on community-based peer reviews to assess the quality of contributions [19]. As the crowd improves their skills, reaching expert levels, they can also get paid to evaluate the submissions. In addition, by componentizing the software development opportunity for malicious code is reduced.

7 Governance and Legal

Corporate governance is the set of processes, customs, policies, laws, and institutions affecting the way an enterprise is directed, administered or controlled.

How are compliance, taxation and labor laws addressed as crowdsourcing is being adopted? What should the crowdsourcing guidelines be when engaging internal, external or hybrid crowds? How do all of the above apply to a global enterprise that coexists in many national/regional jurisdictions? These are some of the questions global enterprises face when considering crowdsourcing of their business functions.

How do enterprises shape their governance guidelines to accommodate for crowdsourcing requirements? Key elements that need to be supported include:

1. Crowd selection process: In certain scenarios, it may be desirable to filter the crowd beforehand, or even selectively invite known experts to drive the crowd contributions based on their expertise and certification levels (e.g. DesignCrowd.com's clients may their favorite designers from around the world while keeping the project open to other designers). What level of screening is required for different crowdsourcing tasks?
2. Intellectual property: To protect both contributors and solution seekers IP rights and processes need to be specified. For example, InnoCentive.com provides a mechanism for companies and crowds to sign an agreement protecting confidential information, as well to specify whether the transfer of IP rights is a requirement of the crowdsourcing task.

3. Certificate of Originality, Legal Ownership and Rights of Property: This is of particular importance to crowdsourcing of software development. How are entities in countries that don't abide by the existing enterprise laws handled?
4. Review process and administration: What level of peer-review, automated analysis and client driven review is appropriate? What kind of appeals process should be in place to support the crowd member? TopCoder, for example, provides an appeal mechanism (explicitly based on fact, rather than matters of opinion), for competitiors to comment on the reviews, as well as file petitions if there was an insufficient data provided.
5. Payment systems: Considering the granularity of the task, how can micropayments and diversity of crowd types (e.g. internal, contractors and external crowds be efficiently handled).

8 Social Factors

How does crowd diversity impact the results? Not only does engaging crowds across geographies and demographics call for careful incentive design, but also for examination of key factors that may drive the successful and high quality contributions.

Jeppesen and Lakhani [20] have examined the winning solvers in high value problems, posted to InnoCentive.com science problem solving contests. They observed and modeled technical and social marginality as strong predictors of success in competition participation. Authors look at and raise potential factors that impact the probability of a crowd member participating in the contents, such as: award value, solution requirements, past problems opened, gender and ethnicity. Similarly, they evaluate to which extent expertise, gender, scientific profile, time invested and interest affect the winning probability.

What does it mean that the crowd is diverse? Moreover, the question arises: do different approaches to producing output in crowdsourcing require different levels of diversity (e.g. competition based versus collaboration based model?).

Oliviera et al [21], provide early insights into challenges of engaging subject matter experts in an open innovation process, and identify the difference between North American and European contexts.

Brabham [22] distills a research agenda along the social aspects from the perspective of the participants, calling for discovery of not only successful crowd members, but also those who are yet to get there. Brabham further identifies some of the barriers to truly global crowdsourcing efforts, such as access to problem-specific skills and technologies.

9 Business Models and Viability

To make crowdsourcing even more appealing, challenge becomes how to deploy it at the minimum cost to the business, while preserving the brand. Costs comprise from staffing and support, marketing, infrastructure and payment services.

Boudreau and Lakhani [23] categorize existing crowdsourcing platforms based on their underlying business model as follows:

1. Integrator platform: where the company integrates the outside innovations and sells the final product to customers - e.g. TopCoder.com, InnoCentive.com.
2. Product platform: where the company enables innovators to build on top of the platform and sell the final products to customers, e.g. GoreTex.
3. Two-sided platform: where the company enables external innovators and customers to interact freely, e.g. Mechanical Turk.

10 Summary

Crowdsourcing has a potential to significantly transform the business processes, by incorporating the knowledge and skills of globally distributed experts to drive business objectives, at shorter cycles and lower cost.

Many interesting and successful examples exist, such as GoldCorp.com, TopCoder.com, Threadless.com, etc. However, to fully adopt this mechanism enterprises, and benefit from appealing value propositions, in terms of reducing the time-to-value, a set of challenges remain, in order for enterprises to retain the brand, achieve high quality contributions, and deploy crowdsourcing at the minimum cost.

In this paper we presented a landscape of existing crowdsourcing applications, targeted to the enterprise domain. We segment crowd into internal, external and hybrid, based on their affiliation with the crowdsourcing firm; and discuss how they drive the selection of incentive and governance structures. Finally, we discuss the effect of engaging socially and technically diverse crowds, and present state of the art in currently deployed business models.

References

1. Vukovic, M., Bartolini, C.: First International Enterprise Crowdsourcing Workshop. In: Daniel, F., Facca, F.M. (eds.) Current Trends in Web Engineering – ICWE 2010 Workshop Proceedings (2010) (in publication)
2. Vukovic, M.: Crowdsourcing for Enterprises. In: International Workshop on Cloud Services, In Conjunction with 7th IEEE International Conference on Web Services (July 2009)
3. Karnin, E., Walach, E., Drory, T.: Crowdsourcing in the Document Processing Practice. In: Proceedings of First Enterprise Crowdsourcing Workshop in conjunction with ICWE 2010 (2010)
4. Lopez, M., Vukovic, M., Laredo, J.: PeopleCloud Service for Enterprise Crowdsourcing. In: International Conference on Services Computing, Miami, Florida (July 2010)
5. Vukovic, M., Lopez, M., Laredo, J.: People cloud for globally integrated enterprise. In: First International Workshop on SOA, Globalization, People, & Work, in conjuction with Seventh International Conference on Service Oriented Computing (2009)
6. Stewart, O., Lubensky, D., Huerta, J.M.: Crowdsourcing participation inequality: a SCOUT model for the enterprise domain. In: Proceedings of the ACM SIGKDD Workshop on Human Computation, HCOMP 2010, Washington DC, July 25, pp. 30–33. ACM, New York (2010)
7. La Vecchia, G., Cisternino, A.: Collaborative workforce, business process crowdsourcing as an alternative of BPO. In: Proceedings of First Enterprise Crowdsourcing Workshop in conjunction with ICWE 2010 (2010)

8. Chen, K.Y., Fine, L., Huberman, B.: Predicting the Future. Information Systems Frontiers 5(1), 47–61 (2005)
9. Surowiecki, J.: The Wisdom of Crowds. Anchor (2005)
10. Archak, N., Sundararajan, A.: Optimal Design of Crowdsourcing Contest. In: Proceedings Thirtieth International Conference on Information Systems (ICIS 2009), Phoenix (2009)
11. Carpenter, H.: – Four Models for Competitive Crowdsourcing, Technical Report (2009), http://spigit.com
12. Kern, R., Thies, H., Bauer, C., Satzger, G.: Quality Assurance for Human-based Electronic Services: A Decision Matrix for Choosing the Right Approach. In: Proceedings of First Enterprise Crowdsourcing Workshop in conjunction with ICWE 2010 (2010)
13. von Law, A.: Input-Agreement: A new Mechanism for Collecting Data Using Human Computation Games. In: Proceedings ACM Conference on Human Factors in Computing Systems, CHI 2009, pp. 1197–1206 (2009)
14. Sorokin, A., Forsyth, D.: Utility data annotation with Amazon Mechanical Turk. In: Proceedings of the Conference on Computer Vision and Pattern Recognition Workshops. IEEE Computer Society, Washington (2008)
15. Snow, R., O'Connor, B., Jurafsky, D., Ng, A.Y.: Cheap and fast, but is it good? Evaluating non-expert annotations for natural language tasks. In: EMNLP 2008: Proceedings of the Conference on Empirical Methods in Natural Language Processing. ACL, Stroudsburg (2008)
16. Sheng, V., Provost, F., Ipeirotis, P.: Get Another Label? Improving Data Quality and Data Mining Using Multiple, Noisy Labelers. In: Proceedings of the Fourteenth International Conference on Knowledge Discovery and Data Mining (KDD) (2008)
17. Kittur, A., Kraut, R.E.: Harnessing the wisdom of crowds in Wikipedia: quality through coordination. In: Shen, W., Yong, J., Yang, Y., Barthès, J.-P.A., Luo, J. (eds.) CSCWD 2007. LNCS, vol. 5236. Springer, Heidelberg (2008)
18. Chen, K., Chang, C., Wu, C., Chang, Y., Lei, C.: Quadrant of euphoria: a crowdsourcing platform for QoE assessment. Network Magazine of Global Internetworking (2010)
19. Lakhani, K., Garbin, D., Lonstein, E.: TopCoder (A): Developing Software through Crowdsourcing. Harvard Business School Case 610-032
20. Jeppesen, L., Lakhani, K.: Marginality and problem solving effectiveness in broadcast search. Organization Science 20 (forthcoming)
21. Oliviera, F., Ramos, I., Santos, L.: Definition of a Crowdsourcing Innovation service for the European SMEs. In: Proceedings of First Enterprise Crowdsourcing Workshop in conjunction with ICWE 2010 (2010)
22. Brabham, D.: Crowdsourcing as a model for problem solving: An introduction and cases. Convergence: The International Journal of Research into New Media Technologies 14(1), 75–90 (2008)
23. Lakhani, K., Boudreau, K.: How to Manage Outside Innovation. MIT Sloan Management Review 50(4)

Analyzing Collaboration in Software Development Processes through Social Networks

Andréa Magalhães Magdaleno[1,3], Cláudia Maria Lima Werner[1], and Renata Mendes de Araujo[2,3]

[1] COPPE/UFRJ – Systems Engineering and Computer Science Department
Zip 21945-970 – Rio de Janeiro – RJ – Brazil – P.O. Box 68511
[2] Department of Applied Informatics – UNIRIO
[3] NP2Tec - Research and Practice Group in Information Technology - UNIRIO
Zip 22290-240 – Rio de Janeiro – RJ – Brazil
{andrea,werner}@cos.ufrj.br, renata.araujo@uniriotec.br

Abstract. Plan-driven, agile or free/open source are software development models that although effective, cannot fully address all the variability of projects and organizations alone. In this work, it is argued that two distinct characteristics of these models – collaboration and discipline – can be the drivers to tailor software development processes to meet projects and organizations needs. In particular, this article focuses on the aspect of collaboration and argues that it can be analyzed through social networks. In this sense, we studied several tools and identified the requirements necessary to explore collaboration, through social networks, in software development processes. These requirements motivated the construction of EvolTrack-SocialNetwork tool.

Keywords: collaboration, social networks, software processes.

1 Introduction

Software organizations are continually challenged by the need to improve the quality of software products. In this context, the assumption that the adopted software development process directly influences the quality of the developed product [1, 2] has motivated many organizations to adopt maturity models, such as CMMI (Capability Maturity Model Integration) [3].

This "plan-driven" development model has been used to support the definition of more predictable and managed software development processes. The success of some free/open source software (FOSS) projects, like Linux and Mozilla, also caught the attention of academia, industry and users due to their capability to produce high quality software, quickly and free [4]. In addition, agile methods are often presented as an alternative to plan-driven development to cope with changes that occur during a development project through shorter development cycles and with a higher level of involvement and participation of the client [5, 6].

Software organizations engage in a wide variety of projects with different characteristics, where plan-driven, agile and FOSS development models, usually

T. Margaria and B. Steffen (Eds.): ISoLA 2010, Part I, LNCS 6415, pp. 435 – 446.

perceived as opponents, complement each other, because each one works better or faces difficulties in some aspects. None of these development models fulfill all requirements of a specific project or organization. Thus, approaches that balance these development models are necessary.

Process tailoring is the act of particularizing a general process description to derive a new process applicable to a specific situation [7]. Pedreira et al. [8] summarize the negative consequences of inappropriate process tailoring in organizations: the project budget, the development time, and the product quality directly depend upon the quality of the software process; a bad software process may involve unnecessary activities that lead to a waste of time and money, or the omission of those activities that are necessary, which may affect the product quality; and inappropriate process tailoring can cause the software process not to comply with the organizational standard process or with international standards such as CMMI [3].

To avoid these risks, we claim that process tailoring should consider organization, project and team contexts, using collaboration and discipline as main drivers [9]. Discipline refers to project control and visibility, while collaboration focuses on people interaction to achieve common goals. These two characteristics are present in plan-driven, agile and FOSS development models, albeit with variations in emphasis and form. Both are complementary and essential in any project, but in different proportions, depending on the project context. Therefore, they need to be balanced.

This paper particularly focuses on the aspect of collaboration and argues that it can be turned explicit using social networks. In this sense, we studied several social network tools and identified the requirements necessary to explore collaboration, through social networks, in software development processes. These requirements motivated the construction of EvolTrack-SocialNetwork tool.

The remainder of this paper is organized as follows. In Section 2, we present the main characteristics of each software development model. Section 3 details the solution focus on balancing collaboration and discipline. Section 4 is dedicated to social networks. Finally, Section 5 concludes the paper.

2 Software Development Models

A software development model is a set of practices recommended for developing software. These practices are organized into a software process that corresponds to "a coherent set of policies, organizational structures, technologies, procedures and artifacts required to design, develop, deploy and maintain a software product" [2].

Plan-driven, agile and FOSS development models have the same goal: to improve software development, but they adopt different approaches. While the plan-driven development seeks for predictability, stability and reliability [3], agile development tries to quickly add value to business and adapt to market, technology and environment changes [6]. Furthermore, in FOSS development, the main objective is to guarantee users' freedom [10].

Plan-driven development is typically exemplified by maturity models, such as CMMI [3], and is characterized by its orientation to planning and emphasis on well-defined processes. The assumptions of agile development, observed in methods such as XP (Extreme Programming) [11] and Scrum [12], can be summarized by the four

values of the Agile Manifesto [5]. The FOSS development can be understood by the bazaar metaphor [4], where projects are collaboratively and transparently developed. In this model, developers work on a voluntary basis, geographically distributed, using the Internet as a communication channel.

Each one with its peculiarities, successes and challenges, these three development models have followed separate paths. Due to differences in vocabulary, misinterpretation and misuse of approaches, they are usually perceived as opponents. However, all of them had, in the last decade, an enormous impact and their perspective for future developments is also promising [13, 14]. As each one represents a universe of development with unique characteristics, research in the area has discussed how to accommodate each of them in order to define development processes that are more effective [15-17].

According to the results obtained through a systematic review, several researchers have investigated the possibility of reconciliation among plan-driven, agile, and FOSS models [18]. In general, the existing proposals [19, 20] involve the comparison and combination of the practices suggested by different models, aiming to produce a new hybrid one. However, the complexity of software development and the variety of existing methods make the task of comparing them, arduous and inaccurate. This kind of software development models combination limits the potential for synergy among them, possibly resulting in an incomplete method, where it is no longer possible to ensure that the resulting process actually has the desired characteristics.

Boehm and Turner [15] proposal suggests risk analysis of the project characteristics as a way to select the project adequate method. This proposal has similarities with our research work, since it considers project characterization. However, it only focuses on agile and plan-driven, without considering FOSS development.

This work argues that more than the combination of practices of different models is necessary. The proposed solution involves software development processes tailoring, by balancing the main distinctive aspects in plan-driven, agile and FOSS models – collaboration and discipline.

3 Collaboration and Discipline

Collaboration can be defined as the group working of two or more people to achieve a common goal. The collaboration is an important factor for software organizations to achieve their goals of productivity, quality and knowledge sharing. In particular, software development is a complex process that involves the collaboration of several people over a period of time to achieve a common goal [1]. Therefore, software development is a typical example of collaborative work [21, 22].

Moreover, the discipline is related to the planning level adopted in software process definition and the rigidity of control employed in process execution. Thus, discipline imposes order and helps to control the work [15].

Both are complementary and essential in any project, but in different proportions, depending on the project characteristics [9]. For a balanced mix between collaboration and discipline, it is necessary to understand how these aspects vary and distinguish the software development models.

Regarding collaboration, we can consider the different levels of formality in communication, coordination, awareness and memory [23]. Regarding discipline, software models vary on emphasis and form of their processes. The plan-driven model is characterized by an emphasis on well-defined and continuously improved processes. Both agile and FOSS development use no description or explicit modeling of the process adopted. Instead, they deal with a set of general principles to guide the development.

For introducing collaboration and discipline into tailored software development processes, it is important to define how to plan and monitor the needed or desired levels of collaboration and discipline. In this sense, some instruments are being considered, such as the Collaboration Maturity Model (CollabMM) [23], social networks [24], and measurement [25].

CollabMM [23] aims to organize a set of practices which can enhance collaboration in business processes. This model describes an evolutionary path in which processes can progressively achieve higher capability on collaboration, according to four maturity levels: Ad-hoc, Planned, Aware and Reflexive. Each level has its own collaboration practices defined according to collaboration support aspects (communication, coordination, group memory and awareness). CollabMM has already been used in a real experience to assist organizations in introducing different levels of collaboration in their business process models [26]. In this work, the role of CollabMM is to act as a framework that defines the collaboration levels and summarizes its main characteristics.

Besides CollabMM, we also need a mechanism that helps to explicit and measure the existing collaboration among people in software development projects. To provide this understanding, social networks [24] appear as a promising path. A social network consists of a finite set of actors and the relationships among them. We can find several works [27, 28] on social network visualization and analysis, which point to social networks potential to explain how the collaboration occurs within a group.

When we explicit collaboration, its visibility increases, so that members of the organization can achieve greater understanding and motivate themselves. Thus, understanding the social networks involved in development projects can help to understand and monitor the level of collaboration in the project.

Discipline can be measured by regulating the level of control of processes. The level of discipline is established through a measurement approach. The need of measuring results comes from the premise that “you can not control what you can not measure”. Measurement is an important mechanism for visibility into a project and helps to raise awareness about ongoing processes.

In particular, this paper focuses on the use of social networks as a mechanism that helps to explicit and measure the existing collaboration among people in software development projects, as discussed in the next section.

4 Social Networks

A network is a graph and consists of a finite set of nodes and edges. In a social network, nodes represent actors and edges correspond to possible relationships among

them [29]. The semantics of the relationship depends on the analysis that will be made in this network. This can be communication, relationship, friendship and so on. In software development, we intend to use the social network to understand the collaboration among team members.

Adapting the approach proposed in [30], we can summarize in four steps the methodology for studying social networks. The first step is to define the purpose of analysis and provide the semantics of nodes and edges of the network. The next step is to collect data to build the social network. This collection can be done manually, using, for instance, questionnaires or facilitated by data mining in the repositories. Then, these data can be manipulated for viewing or analysis.

Next, in the social networks visualization step, the visual representation of information is adopted to reduce the cognitive overload of the user and to facilitate understanding and exploration of data through graphs. The social networks visualization allows the observation of facts and knowledge extraction.

Finally, the last step is the analysis of social networks, which uses the concepts of graph theory to describe, understand and explain the interaction and social organization of a group. This analysis seeks to understand the relationships among people or groups through its properties. These properties [29] were detailed in a previous study [31], which also identified those (i.e., degree centrality, betweenness centrality, closeness centrality and network density) with the greatest potential to explain collaboration.

Based on the interpretation and combination of social network analysis properties, Santos et al. [31] suggest a set of collaboration characteristics, organized regarding CollabMM maturity levels [23]. These first characteristics focused on the aspect of coordination. The main idea is that social network properties can be associated to the different collaboration levels suggested by CollabMM.

4.1 Requirements for Social Networks Tools

In a previous study [32], we analyzed 10 social network tools: one shareware (UCINET[1]), one open source (Pajek [33]), and 8 academic (Ariadne [34], Augur [35], MiSoN [36], OssNetwork [37], RaisAware [38], Sargas [39], SVNNAT [40], and Visone [41]).

This study showed that some of the identified tools already provide an extensive set of generic algorithms that can be readily used to calculate social network properties. However, they do not engage in analysis dedicated specifically to the collaboration. In addition, most visualization tools are not actually available or have significant limitations [32].

Considering the analysis of contributions and limitations of these tools, the existing proposals for social networks analysis in software development, and the objectives of this research work, we gathered a list of 15 requirements that a social network tool must satisfy [32]. These requirements were classified into three categories: mining (REQM), visualization (REQV) and analysis (REQA). A few examples of these requirements are presented in Table 1.

[1] UCINet Site: http://www.analytictech.com/ucinet/ucinet.htm

Table 1. Examples of social network tools requirements

Name	Description
REQM3	The system must be able to mine data from different sources of information for software development projects: a repository of configuration management, source code, discussion forum, and e-mail list.
REQV10	The system must provide the visualization of network evolution over time.
REQA11	The system must calculate the properties of social network analysis.

The studied social networks tools were analyzed in accordance with this list [32], as partially presented in Table 2. Fields filled with ✔ indicate that the requirement is fully covered by the tool, while ✘ indicate that the requirement is not met by the tool. The fact that a requirement is not met by a particular tool can only mean that this specific requirement may be irrelevant for its purpose. Besides, these requirements not form a complete nor necessarily sufficient list, they serve as a guide for developing a tool that intends to be comprehensive.

Table 2. Tools x requirements

Requirement	Ariadne	Augur	MiSoN	OssNetwork	Pajek	RaisAware	Sargas	SVNNAT	UCINET	Visone
REQM3	✘	✘	✘	✔	✘	✘	✔	✘	✘	✘
REQV10	✔	✘	✘	✔	✘	✘	✘	✘	✘	✘
REQA11	✘	✘	✘	✔	✔	✔	✔	✘	✔	✔

After this analysis, we concluded that none of the analyzed social network tools met all requirements. Thus, there is still room to propose other tools that support in a more adequate manner the need for collaboration analysis and bring new contributions, through the implementation of these requirements. This motivated the creation of a new tool - called EvolTrack-SocialNetwork.

4.2 EvolTrack-SocialNetwork

EvolTrack-SocialNetwork is an extension of EvolTrack tool [42], which was developed by the Software Reuse Group at COPPE/UFRJ in Brazil. EvolTrack is a software visualization tool that provides a time based approach to observe the

emerging design at different moments during the development life cycle. Basically, it periodically extracts project information from a specific data source and then, after performing some pre-processing and transformation, presents the corresponding software design for that project period of time.

EvolTrack was chosen as the starting point for building EvolTrack-SocialNetwork, because it offers an initial infrastructure for data mining, some visualization features and functionality for analyzing metrics, and has been developed by the same research group in which this work is being developed.

After this choice, a study to examine EvolTrack feasibility for use in real scenarios was planned and conducted, using seven FOSS projects [42]. These projects were chosen because they publish their development artifacts, including source code, freely over the Internet. Thus, they represent an opportunity for research due to its diversity, complexity, representativeness, and ease of access.

As a result, this study showed the feasibility of using EvolTrack and highlighted some scalability limitations in relation to the display of models of very large projects [42]. This problem was recently solved as part of another ongoing research project that aimed to expand the capabilities of EvolTrack visualization [43].

After this assessment of the feasibility of using EvolTrack, the design of EvolTrack-SocialNetwork architecture was started. This architecture is composed by three main modules: mining, visualization and analysis (Fig. 1). All of these modules are based on a social network meta-model. Moreover, since we reused EvolTrack infrastructure, its components are also used.

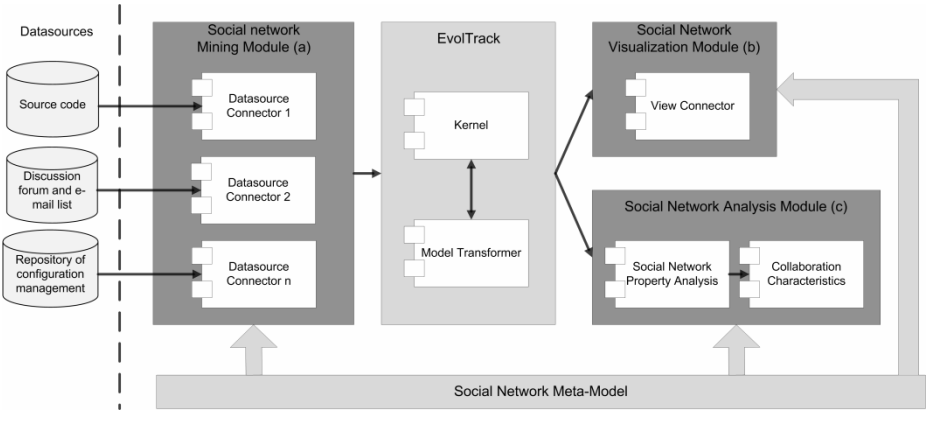


Fig. 1. EvolTrack-SocialNetwork Architecture Overview

Originally, EvolTrack uses the UML meta-model. Although this meta-model is suitable for processing the information of technical networks, it does not fully address the needs of information for socio-technical and social networks. Thus, this meta-model needs to be extended in EvolTrack-SocialNetwork. The choice of a social network meta-model should be based on the analysis you want to accomplish with the social network.

The mining module (Fig. 1a) receives the information extracted by datasources connectors. EvolTrack already has connectors for configuration management tools

(like SVN) and development environments (such as Eclipse). In addition, EvolTrack provides the necessary infrastructure to facilitate the creation of new connectors. In this case, the datasource connectors that need to be constructed in EvolTrack-SocialNetwork are: e-mail list and discussion forum.

The visualization module (Fig. 1b) receives the information processed by the kernel and focuses on offering technical, socio-technical and social networks [34] on project software development. The first one is already implemented in EvolTrack because the UML view connector already expresses by the class diagram, exactly what one wants to represent in the technical network, where the nodes are classes and edges represent the dependency relationships among them. EvolTrack also offers the presentation of the temporal evolution of the network.

Finally, the analysis module (Fig. 1c) is concerned about the need to analyze the properties of social networks [31]. Although the EvolTrack does not calculate these properties, it is able to display them and follow the evolution of metrics in general [42].

Currently, EvolTrack-SocialNetwork tool is under development and we expect that it can contribute to provide information that will help both the development team and project manager to understand, reflect and interfere in the work being done.

4.3 Scenario of Collaboration Information Use

Suppose a situation in which a software development project, called CDSOFT, involves a geographically distributed team and it aims to create an innovative product. Due to product novelty, it also has volatile requirements that change frequently. Because of this requirements instability, the documentation becomes obsolete quickly and the need for sharing tacit knowledge among team members is intensified. In this scenario, CDSOFT project would require a high level of collaboration.

Therefore, considering CollabMM maturity levels [23], this project is planned to accomplish reflexive level. CollabMM defines the reflexive level as the one where processes are designed to provide self-understanding, identifying the relevance of the results which had been produced and sharing this knowledge inside the organization. Processes must be formally concluded and their results communicated. Lessons learned can be captured; strengths and weaknesses are analyzed; successes and challenges are shared; ideas for future improvements are collected; and workgroup results are published and celebrated. Group members are aware of the way in which the group collaborates during process execution, while process tacit knowledge is shared, thereby enhancing group memory [23].

The collaboration characteristic of distributed coordination represents this level. This characteristic can be perceived by the absence of intermediate and central nodes in the network, because the relationships between the nodes tend to be equally distributed. The network density is considered high and may reach its maximum, which also represents the maximum degree of collaboration. Therefore, it is expected that CDSOFT project has a network similar to the one presented in Fig. 2a.

However, during project execution, one can notice that the actual project social network is looking like the one in Fig. 2b. In this centralized network, it can be observed a strong leadership of one single node that is controlling tasks and information flow. Based on this information, it is possible to use CollabMM to plan which collaboration practices (such as communication plan and collaboration awareness) can be included to make this project more collaborative.

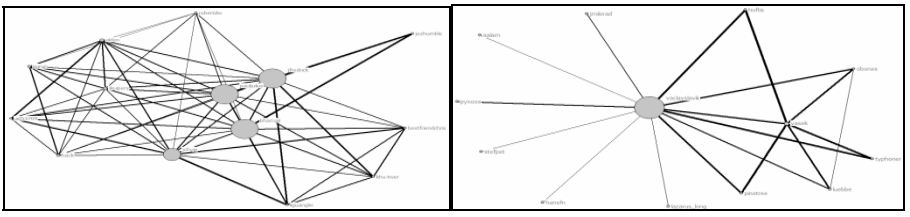


Fig. 2. Current and Planned Social Networks

Even in such a simple example, we can notice that this kind of information is helpful to understand, reflect and interfere in the work being done. The combination of CollabMM and social networks instruments can provide the project manager with useful information to make decisions about the future of the project and enhance collaboration.

5 Conclusion

In this paper, we discussed that social networks, obtained as a result of interactions in software development, can provide useful information for understanding the collaboration among development team members. Our aim is to contribute to research related to the understanding of collaboration in different development models – plan-driven, agile, and FOSS - arguing that the understanding of collaboration can be a way to promote balance between these different approaches. Therefore, this information can be used in process tailoring.

Despite being one of the main tasks to be executed by the project manager, process tailoring is not simple. It requires pondering many factors and evaluating a large set of constraints. Due to this complexity, the manager is not usually able to evaluate all available combinations and chooses a process in an ad-hoc manner, based on his/her own experience, possibly selecting one that is not the best alternative for the current project.

In order to facilitate process tailoring, it is possible to support the project manager by automating some of the steps to solve the problem, possibly reducing the effort required to execute this activity and improving the quality and adequacy of the obtained process [44, 45]. This decision support environment can help in the selection of an appropriate process for a software project according to the best balance between collaboration and discipline. In fact, this possibility is being explored as an optimization-based problem, which uses collaboration and discipline as utility functions to maximize or minimize them [44, 45].

Acknowledgments

This work is partially funded by CNPq (under processes nº. 142006/2008-4 and 310776/2009-0) and is part of Programa Institutos Nacionais de Ciência e Tecnologia, supported by Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq/MCT) (under contract nº. 557.128/2009-9) and by Fundação de Amparo à Pesquisa do Estado do Rio de Janeiro (FAPERJ) (under contract nº. E-26/170028/2008).

References

1. Cugola, G., Ghezzi, C.: Software processes: A retrospective and a path to the future. Software Process Improvement and Practice (SPIP) Journal 4, 101–123 (1998)
2. Fuggetta, A.: Software process: a roadmap. In: Proceedings of the Conference on The Future of Software Engineering, pp. 25–34. ACM, Limerick (2000)
3. Chrissis, M.B., Konrad, M., Shrum, S.: CMMI: Guidelines for Process Integration and Product Improvement. Addison-Wesley, Boston (2006)
4. Raymond, E.S.: The Cathedral & the Bazaar. O'Reilly Media, Sebastopol (2001)
5. Beck, K., et al.: Manifesto for Agile Software Development, `http://agilemanifesto.org/`
6. Cockburn, A.: Agile Software Development. Addison-Wesley, Boston (2001)
7. Ginsberg, M., Quinn, L.: Process Tailoring and the Software Capability Maturity Model. SEI-CMU (1995)
8. Pedreira, O., Piattini, M., Luaces, M.R., Brisaboa, N.R.: A systematic review of software process tailoring. SIGSOFT Software Engineering Notes 32, 1–6 (2007)
9. Magdaleno, A.M.: Balancing Collaboration and Discipline in Software Development Processes. In: Doctoral Symposium of International Conference on Software Engineering (ICSE), pp. 331–332. ACM/IEEE, Cape Town, South Africa (2010)
10. FSF: The Free Software Definition, `http://www.gnu.org/philosophy/free-sw.html`
11. Beck, K.: Extreme Programming Explained: Embrace Change. Addison-Wesley, Boston (1999)
12. Schwaber, K.: Agile Project Management with Scrum. Microsoft Press, Washington (2004)
13. Ebert, C.: Open Source Drives Innovation. IEEE Software 24, 105–109 (2007)
14. Theunissen, M., Kourie, D.G., Boake, A.: Corporate-, Agile- and Open Source Software Development: A Witch's Brew or An Elixir of Life? In: Meyer, B., Nawrocki, J.R., Walter, B. (eds.) CEE-SET 2007. LNCS, vol. 5082, pp. 84–95. Springer, Heidelberg (2008)
15. Boehm, B., Turner, R.: Balancing Agility and Discipline: A Guide for the Perplexed. Addison-Wesley, Boston (2003)
16. Glazer, H., Dalton, J., Anderson, D., Konrad, M., Shrum, S.: CMMI or Agile: Why Not Embrace Both! SEI-CMU (2008)
17. Warsta, J., Abrahamsson, P.: Is Open Source Software Development Essentially an Agile Method? In: Proceedings of the Workshop on Open Source Software Development, Portland, OR, USA, pp. 143–147 (2003)
18. Magdaleno, A.M., Werner, C.M.L., Araujo, R.M.D.: Revisão Quasi-Sistemática da Literatura: Conciliação de processos de desenvolvimento de software. PESC-COPPE, Rio de Janeiro (2009) (In portuguese)
19. Fritzsche, M., Keil, P.: Agile Methods and CMMI: Compatibility or Conflict? e-Informatica Software Engineering Journal 1, 9–26 (2007)
20. Paulk, M.: Extreme programming from a CMM perspective. IEEE Software 18, 19–26 (2001)
21. DeMarco, T., Lister, T.: Peopleware: Productive Projects and Teams. Dorset House, New York (1999)
22. Herbsleb, J.D., Paulish, D.J., Bass, M.: Global software development at siemens: experience from nine projects. In: Proceedings of the 27th international conference on Software engineering, pp. 524–533. ACM Press, St. Louis (2005)

23. Magdaleno, A.M., Araujo, R.M.D., Borges, M.R.S.: A Maturity Model to Promote Collaboration in Business Processes. International Journal of Business Process Integration and Management (IJBPIM) 4, 111–123 (2009)
24. Barabasi, A.L.: Linked: How Everything Is Connected to Everything Else and What It Means for Business, Science, and Everyday Life. Plume, Cambridge (2003)
25. McGarry, J., Card, D., Jones, C., Layman, B., Clark, E., Dean, J., Hall, F.: Practical Software Measurement: Objective Information for Decision Makers. Addison-Wesley Professional, Reading (2001)
26. Magdaleno, A.M., Cappelli, C., Baiao, F.A., Santoro, F.M., Araujo, R.: Towards Collaboration Maturity in Business Processes: An Exploratory Study in Oil Production Processes. Information Systems Management (ISM) 25, 302–318 (2008)
27. Gao, Y., Freeh, V., Madey, G.: Analysis and Modeling of Open Source Software Community. In: Computational Analysis of Social and Organizational Systems (CASOS), Pittsburgh, PA, USA, pp. 1–4 (2003)
28. Madey, G., Freeh, V., Tynan, R.: The open source software development phenomenon: An analysis based on social network theory, Dallas, TX, USA, pp. 1806–1813 (2002)
29. Wasserman, S., Faust, K.: Social Network Analysis: Methods and Applications. Cambridge University Press, Cambridge (1994)
30. Cross, R., Andrew, P., Cross, R.: The Hidden Power of Social Networks: Understanding How Work Really Gets Done in Organizations. Harvard Business School Press, Boston (2004)
31. Santos, T.A.L., Araujo, R.M.D., Magdaleno, A.M.: Identifying Collaboration Patterns in Software Development Social Networks. Infocomp - Journal of Computer Science - Special Issue, 51–60 (2010)
32. Magdaleno, A.M., Werner, C.M.L., Araujo, R.M.D.: Estudo de Ferramentas de Mineração, Visualização e Análise de Redes Sociais. In: PESC-COPPE, Rio de Janeiro, RJ, Brasil (2010) (in portuguese)
33. Nooy, W.D., Mrvar, A., Batagelj, V.: Exploratory Social Network Analysis with Pajek. Cambridge University Press, Cambridge (2005)
34. Trainer, E., Quirk, S., Souza, C.D., Redmiles, D.: Bridging the gap between technical and social dependencies with Ariadne, pp. 26–30. ACM, San Diego (2005)
35. Souza, C.D., Froehlich, J., Dourish, P.: Seeking the source: software source code as a social and technical artifact. In: Proceedings of the International ACM SIGGROUP Conference on Supporting Group Work, pp. 197–206. ACM, Sanibel Island (2005)
36. Aalst, W., Reijers, H.A., Song, M.: Discovering Social Networks from Event Logs. In: Computer Supported Cooperative Work (CSCW), vol. 14, pp. 549–593 (2005)
37. Balieiro, M.A., Souza Jr., S.D., Pereira, L.P., de Souza, C.D.: OSSNetwork: Um Ambiente para Estudo de Comunidades de Software Livre usando Redes Sociais. In: Experimental Software Engineering Latin America Workshop, São Paulo, SP, Brasil, pp. 33–42 (2007) (in Protuguese)
38. Costa, J.M.R., Feitosa, R.M., de Souza, C.D.: RaisAware: Uma ferramenta de auxílio à engenharia de software colaborativa baseada em análises de dependências, Vitória, ES, Brasil, pp. 254–264 (2008) (in Protuguese)
39. Sousa Junior, S.F., Balieiro, M.A., de Souza, C.D.: Análise Multidimensional de Redes Sociais de Projetos de Software Livre. In: Anais do Simpósio Brasileiro de Sistemas Colaborativos (SBSC), Vitória, ES, Brasil, pp. 23–33 (2008) (in Protuguese)
40. Schwind, M., Wegmann, C.: SVNNAT: Measuring Collaboration in Software Development Networks, pp. 97–104. IEEE, Washington (2008)

41. Brandes, U., Wagner, D.: Visone - Analysis and visualization of social networks. In: Graph Drawing Software, pp. 321–340. Springer, Heidelberg (2003)
42. Cepeda, R.D.S.V., Magdaleno, A.M., Murta, L.G.P., Werner, C.: EvolTrack: Improving Design Evolution Awareness in Software Development. Journal of the Brazilian Computer Society (JBCS) (to appear, 2010)
43. Silva, M.A.: IAVEMS: Infraestrutura de Apoio à Visualização da Evolução de Métricas de Software (2010) (in Portuguese)
44. Magdaleno, A.M., Barros, M.D.O., Werner, C.M.L., Araujo, R.M.D.: Formulando a Adaptação de Processos de Desenvolvimento de Software como um Problema de Otimização, Salvador, BA, Brasil (to appear, 2010) (in Portuguese)
45. Magdaleno, A.M.: An optimization-based approach to software development process tailoring. PhD Track - International Symposium on Search Based Software Engineering (SSBSE), Benevento, Italy (to appear, 2010)

A Web-Based Framework for Collaborative Innovation

Donald Cowan[1,4], Paulo Alencar[1,4] Fred McGarry[3], Carlos Lucena[3,5], and Ingrid Nunes[3,5]

[1] David R. Cheriton School of Computer Science and Computer Systems Group
University of Waterloo, Waterloo, Ontario Canada
{dcowan,palencar}@cs.uwaterloo.ca
[2] Centre for Community Mapping, Waterloo, Ontario Canada
mcgarry@comap.ca
[3] Department of Informatics and Software Engineering Laboratory
Pontifical Catholic University of Rio de Janeiro, Rio de Janeiro, Brazil
{lucena,ionunes}@inf.puc-rio.br
[4] The Waterloo Innitiative on Web Science
[5] Brazilian Institute for Web Science Research

Abstract. Twenty-first century global change in most sectors is challenging our use and management of resources. For a community to adapt to this systemic change, while maintaining and even enhancing its economy and quality of life, the World Economic Forum has recognized the need for new approaches to enable collaborative innovation (CI) and related action among both the leadership and concerned members of the community. Many see the web as an approach to CI and as a new form of creativity machine that can augment our intelligence.

This paper outlines the concepts of an approach to CI based on asset mapping and how it has been supported through a web-based technological framework that requires both communication and operations on the asset map. Based on the experience using the framework in designing and building over 50 systems that incorporate asset-mapping CI, it is clear that CI takes many forms. We illustrate some of these forms through specific examples in environment, cultural heritage, socio-economic development and planning. We conclude that it is not possible to build a single set of tools to support CI and that the users need access to meta-tools and frameworks to implement tailored systems supporting CI directly rather than relying on people with in-depth knowledge of the technologies. WIDE is an example of the type of meta-tools that are needed. Lessons learned from WIDE are being applied in the creation of WIDE 2.0.

Keywords: collaborative innovation; asset-mapping; web-based framework; meta-tools.

1 Introduction

Twenty-first century systemic global change in health, water, environment, energy, business, governance and socio-economic structures is challenging our

T. Margaria and B. Steffen (Eds.): ISoLA 2010, Part I, LNCS 6415, pp. 447–461, 2010.

communities and us as individuals. For a community to be resilient and adaptable in the face of these disruptive transformations, while maintaining and even enhancing our economy and quality of life, new creative approaches are needed to enable intelligent collaborative innovation (CI) and related action. Such action must occur among all sectors of society including government, non-governmental organizations (NGOs), business, community leadership and concerned community members.

During the last two decades we have seen the proliferation of web-based information and communication technology (ICT) tools such as shared databases, mapping, wikis, blogs, on-line communities and social networks that support connection and communication among disparate groups and individuals. Can we harness the power of the web and this new collaboration paradigm to augment our intelligence as a new form of creativity machine [1] and support CI to address the problems arising from these systemic changes? Currently the existing tools do not integrate well and thus limit our ability to innovate collaboratively over the web. Thus the goal of our research is two-fold:

- to develop operational software systems to support CI in multiple domains in order to understand
  - how CI can be implemented effectively using the web and
  - how CI can be made a self-sustaining activity
- and to identify and develop web-based software approaches
  - for meta-tools or frameworks that significantly simplify the construction and evolution of CI systems and
  - that can provide for the simple integration of existing tools into CI frameworks.

As an example, the World Economic Forum (WEF) [2] has recognized the need for an effective approach to CI in the aftermath of the global economic crisis as local and international communities are becoming more and more interdependent. The WEF has recommended that communities both geographic and virtual re-examine their strategies based on new evolving networks and forms of collaboration. Thus public-private partnerships and smaller-scale region or city-driven initiatives must move away from one-size fits all arrangements and re-design and rebuild their structures and processes related to both local and global community interactions, resource exploitation and governance.

The focus of this paper is to examine experiences with CI within the context of the new modes of communication that have arisen through the development of the Internet, Web and tools such as mapping and social networks and to propose new approaches to meta-tools and frameworks that would bring substantive improvement to web-based CI. We use examples of web-based systems for CI in specific fields such as environment, cultural heritage, tourism, socio-economic development and planning to demonstrate our current thinking.

We observe in the paper that Web-based CI involves both communication and operations over a dynamic repository of assets and needs not only communication

tools but tools that support the varied operations on this repository. Such tools are specific to the CI tasks and therefore must created for each Web-based CI system. Because the need for the tools is provided by the group involved in the CI, they need to specify the tools and hopefully generate them from the specification. If they have to involve technologists such as programmers and GIS experts then the spontaneity of the collaboration may be impacted negatively.

The Web Informatics Development Environment (WIDE) was designed to meet the need just delineated. WIDE is a set of technologies, processes and meta-tools designed and developed by the Computer Systems Group at the University of Waterloo (UWCSG - http://csg.uwaterloo.ca/) to support the creation of meta-tools or frameworks that can be easily completed by members of the collaborating group. Once the dynamic asset repository is established members of the group can specify the required tools to support specific collaborative activity. Over 50 web-based information infrastructures supporting CI in many forms have been developed by the Centre for Community Mapping (COMAP - http://comap.ca/) and UWCSG based on the WIDE technologies.

2 Understanding CI

CI refers to a process that involves assembling a team of people to explore and act upon change in an idea or situation [3]. The span of collaboration is a virtual community of practice or interest, or a geographic community [4]. The individuals that compose the team can represent themselves, different departments in a single organization or different organizations. A team often works in a mediated environment where responsibility for actions can be devolved to specific team members. These responsibilities can be specified through one or more contracts among individuals and organizations.

CI teams have a collective vision and wish to work together by sharing ideas, information and work. Team members share directly rather than through a hierarchy, although each member may represent the views of the hierarchy to which he/she belongs. Any collaboration must have four essential elements, which are:

1. sound ethical principles and trust;
2. self-organization;
3. universally accessible knowledge; and
4. honest and transparent operation.

Organized CI has been recognized as early as Benjamin Franklin's "Junto" organization. However, today most work of this type, relies on modern information and communications technology (ICT) such as the Internet, e-mail, the Web and more recently social networks.

What else is needed to make CI effective? CI requires shared knowledge of the resources or assets that are available within an application domain or context and then using that understanding to implement actions such as operational and structural decisions. Thus CI can be viewed as a two-step process where members of a geographic or virtual community (community of practice):

1. produce an inventory of assets and share this data across one or more communities; and
2. collaborate and act on that asset knowledge to:
   (a) recognize additional "undiscovered" assets;
   (b) produce value and change; and
   (c) create new assets related to the change.

Basically the participants share knowledge and add value to that shared knowledge through a collaborative set of tools that operate under various constraints such as access, time and mobility.

In many fields, the concept of creating an asset inventory is called "asset mapping" [5] and the new value and assets produced from the collaboration are often called emergent properties. Most work in this field to date has focused on "static" asset mapping and has only begun to investigate the methodology to keep the asset inventory current or "dynamic" and in taking action over the growing asset base. Further research is needed into an organized approach to on-going collaboration, dynamic asset mapping, and value production, although this lack of a theory of CI should not stop its use. In this paper we outline a novel asset-mapping driven approach to web-based CI based on our experience in designing and implementing over 50 such systems.

3 CI, Dynamic Asset Mapping and the Web

There are many facets to CI using ICT and based on a dynamic asset mapping approach. In this section we analyze this phenomenon and try to understand its constituents and related properties. Basically the participants share knowledge and add value to that shared knowledge through a collaborative set of tools that can be constrained under various properties.

3.1 Constituents of CI and Dynamic Asset Mapping

Shared knowledge. There must be a collection of shared information or knowledge, often called a database, knowledge base, inventory or asset map, to provide a basis for collaboration that takes advantage of intelligent software agents to manage redundancy [6]. For example, an environmental group interested in tracking and eliminating invasive species in a region would likely start with a database of all such known species in the geographic area and the protocols associated with their identification and remediation.

The Collaborative Canvas. Information about these assets and input to and results of the collaboration can be presented through shared collaborative canvases or interfaces. The shared canvas can provide many different tools such as a data report, input to databases, e-mail, audio, video, pictures, text, maps (geomatics), short messages (Twitter tweets), blogs, and wikis to name some examples. Thus, a member of a collaborative can view the current state of the

assets by reading a report, viewing a presentation on a map or reading a blog or wiki. These canvases can allow members to provide new information to the set of assets through an input form, by allowing a drawing on a map or diagram, or through commentary in a blog or wiki. For example, a report showing a list of sightings of invasive species can indicate the prevalence of the species, while a map showing the geographic location of the sightings provides even more valuable information since it indicates the geographic extent of the unwanted species invasion. The map can also be used to allow anyone providing information about an invasive species sighting to provide its geographic location by placing an electronic pin in the map. Similarly an active collaborator to a historical event collaboration could provide a video of a battle re-enactment which has been attached to a map where the location and extent of the battle could be specified.

Access control model. Collaboration requires that a group forms around an idea or situation with the objective of working together. The group is self-limiting by expertise or interest although it may grow or change in composition as the collaboration forms and changes, and new expertise is needed or more individuals become interested. By its nature collaboration is not completely open. Therefore there must be moderators who manage the group composition and delegate authority to members of the group related to responsibilities. These moderators must be given a set of tools not only to enable the collaboration but to admit participants with responsibilities, so-called transactional access controls. Such access controls can be role-based [7] or use other confidentially models [8] that can control participation. Participants could be allowed a subset of operations on the asset base such as read, write, update, or write/update with history log. These same participants could also be limited in the collaborative canvas tools that they can use or the portion of the asset base they can see through an interface such as a map. For example, experts in invasive species could use a wiki or blog to discuss issues around protocols for identification and remediation, whereas laymen could read the content of the wiki or blog, but would not be able to augment the expert opinion.

Creative Social Networks and On-line Communities. A creative social network or on-line community can be viewed as a mediated social network or on-line community with a set of purposes derived from its relationship to members of a geographic community or community of interest from which the creative social network is invoked. In other words, a creative social network/on-line community is built for one or more purposes. In social networks communication and related assets belong to the individual who created them, while in an on-line community the information belong to community to which the individual community belongs.

Social networks/on-line communities are primarily designed to share without being concerned about the nature of the content. Both types of networks can use tools from the collaborative canvas that allow sharing and collaboration over the asset base with responsibilities usually delegated through role-based

access controls. There can be many different types of tools that are used as components of the both types of social networks as already described including maps (geomatics), input forms, reports, text, video, pictures, audio, wikis, blogs, and asset repositories.

Social networks/on-line communities prompt connection and communication to enhance opportunities for self-organization within virtual communities of interest, practice and geography that have shared goals, protocols and applications. Such communities can gel around subject matter in searchable shared spaces such as forums, wikis and geography (maps) rather than organizational hierarchies. They can also form around groups where individual members find and connect with each other through shared background or knowledge based on user profiles, private messaging, groups, contributed content and Wiki linkage, notification, expert search and other creative and social networks.

Tags, social bookmarks, and other social networking tools can help bring order to the avalanche of information that is involved in forming a creative network and managing the output from the collaboration.

3.2 Properties of Dynamic Asset Mapping CI

Collaboration in Time. Collaboration can occur asynchronously or synchronously (real-time). Asynchronous collaboration is more common because contributors do not have to agree on a time to meet. The collaborators can contribute through e-mail, maps, wikis, blogs, Web forms and other similar mechanisms. The example in a previous paragraph about posting sightings of invasive species on a map is being performed as an asynchronous independent action. It does not require the participation of anyone else.

Synchronous or real-time collaboration requires the presence of two or more parties to the collaboration. For example if a land developer, local government and concerned citizen's group are negotiating (collaborating) over the site and extent of a commercial mall, then they could share a common map for the discussion. Of course all the collaborators could be in the same room or share the map and associated communication channels over the Web.

Mobility. Mobility adds a new dimension to collaboration as now the collaboration and the creation of value can happen while participants are on the move. When locating an invasive species, an individual can locate it on a paper map which can then be transcribed to a Web-based map when he/she returns to their desktop or laptop computer. Instead a smartphone with its global-positioning system (GPS) can be used to record the location on a map where the map might also be shown on the phone. Thus mobility has the potential to provide more accuracy and immediacy as the the data can be captured and reported upon in real-time.

Adding Value. Once the collaboration begins the active collaborators through working together using the shared assets and collaborative canvas add value since they create more information and knowledge or assets, which are usually

added to the asset base. These additions are called emergent properties. Rich emergent properties often arise when diverse groups form communities of practice. Returning to the environmental group that is interested in invasive species, individuals could join the collaboration and pinpoint the location of sightings of invasive species thereby adding emergent properties to the asset base by giving an indication of a species' progress across the landscape.

4 Implementing CI Based on Dynamic Asset Mapping

The UWCSG and COMAP have undertaken joint research in developing technologies, processes and meta-tools as approaches to building web-based information systems to support CI based on dynamic asset mapping. This work has resulted in the Web Informatics Development Environment (WIDE) technologies and toolkit of meta-tools, and over 50 web-based systems many of which support different forms of CI.

Collaboration and CI require communication as described in [9], but CI requires far more. Specifically CI must contain tools that support the construction, manipulation, analysis and updating of the dynamic asset map by the members of the collaborating group while integrating communication, since both are essential to CI. Typically these tools support the creation of input forms and reports that also integrate components from the collaborative canvas as well as social networks and on-line communities

Presently the creation and integration of such tools requires the intervention of technologists such as programmers or GIS experts. The addition of this gatekeeper to the collaboration team can cripple the collaboration by subverting the spontaneity of the interactions. As mentioned in [9], time is a scarce resource and any activity that slows down collaboration will likely cause many of the collaborators to lose interest and move on to other more rewarding projects.

Because of this requirement to integrate known and new web services, an efficient approach to building and testing such complex technologies with direct action from collaboration participants is needed. The toolkit based on the WIDE technologies offers approaches that are simple to use and to change - programmers are rarely needed. In the WIDE context, the user specifies the required web-based system using a a wizard-based approach and then the specification is used to generate the system. Based on our observations to date this approach allows the collaboration group to develop web-based information systems about 10 times faster than through more traditional methods. We are continuing to improve the WIDE technologies to make them easier to specify systems.

Collaboration should be capable of being mediated in that many collaborations require that all the participants be identified. This condition is not meant to be onerous, but rather to track the origin of ideas and how they evolve. Our view is that anyone can join a collaboration but they should be known. However, the amount of identification (including none) can be left to those who establish the initial collaboration group. In addition we believe the content of the collaboration, that is both the communication and the dynamic asset map and the

"publication" of associated results should belong to and be under the control of the collaboration group. Further the results should not be available for mining and sale outside the group of collaborators without the group's specific permission. Many current tools do not support adequate mediation and ownership of the results often resides with the supplier(s) of the tool(s).

Participating organizations are provided with the ability to create mediated social networks, on-line communities and other communication services that support self-organization among participants by promoting opportunities to share knowledge with other participants with common interests or goals. By combining dynamic asset mapping with social network services and online communities, communities of practice can collectively communicate to discern opportunities for CI.

The WIDE technologies and toolkit is a set of processes and meta-tools that have been conceived to solve the problem of allowing the collaboration team to construct and integrate its own tools for communication and for operation on the dynamic asset map. The meta-tools in the WIDE include services related to temporal and spatial (mapping) data, role-based access controls, reporting, document management, creative social networks and other collaboration tools, that is WIDE implements tools that support the concepts in the previous section. The WIDE toolkit is designed to "wrap" and incorporate available tools that meet the previously mentioned criteria. This research program related to the WIDE technologies has a number of goals namely to:

1. develop data models and corresponding databases for dynamic asset maps in different domains;
2. create meta-tools to simplify the production of interfaces, access controls and creative social networks and online communities;
3. create interface and social network frameworks to asset maps to support powerful and purposeful collaboration;
4. create interface frameworks to asset maps to assist with collaborative and dynamic asset map updating;
5. create interface frameworks or "wrappers" that can be used with existing tools that support best practices so that they can be integrated into the WIDE approach;
6. create interface frameworks to interrelate information maintained by different communities of practice to enable synergistic benefits; and
7. produce meta-tools that can be used and maintained by the collaborators.

Currently COMAP and UWCSG and the users completely engage during the entire specification, design and implementation cycle and use an iterative approach within the context of the WIDE technologies to create complex web services. The approach is iterative in that once users operate a version of the system, they may actually change the specification on the fly, thus affecting further design and implementation. The goal of the next version of the WIDE technologies is to provide the approaches and meta-tools to allow the user to specify and generate the applications that implement the social networks/on-line communities and all the tools that operate on the dynamic asset map.

5 CI Examples Based on Dynamic Asset Mapping

Different forms of CI depend on factors such as community scope, degree of cohesion among community members, and type of knowledge exchanged and variations on these factors can lead to different types of communities [10]. The WIDE technologies have been used to support many different forms of CI with varying sizes of geographic and virtual communities and different degrees of community coherence based on common goals. The different forms of CI supported by the WIDE technologies include:

1. Geomatics (mapping);
2. Dynamic asset-mapping for geographic communities;
3. Dynamic asset-mapping for communities of practice;
4. Real-time synchronous web-based geomatics services;
5. Laymen as sensors;
6. General public participation; and
7. Mediated communication and social networks.

Some specific projects and related approaches are described in this section.

5.1 CI Based Geomatics (Mapping)

CI based on geomatics consists of a common mapping canvas where communities can collaboratively sustain discussions and communications relating to evolving spatial issues and also use applications based on standard protocols for data capture, analysis and reporting. CI based on geomatics services were first used extensively in the Stewardship Tracking System (STS) where the mapping canvas consists of satellite and highly resolute airphotos with thematic data.

The STS was devised to address the need for conservation planning for retention of ecosystem functions in the Southern Ontario landscape. The STS is a system which enables the tracking of restoration projects (e.g. landscape elements such as woodlots, streams, wetlands, and prairie) and provides for adaptive management amongst the conservation community of practice. A project technical committee comprised of over a dozen non-government and government organizations oversaw the development of the STS as a concurrent iterative design process, informed by widely dispersed workshops and web-forums.

The STS project is a web-based set of applications and database that accesses spatial data and information in real-time from distributed sources over the Internet. The STS was planned to permit the Southern Ontario conservation community participants to work collaboratively by:

- entering spatial (polygon) and tabular data, photos and documents about their ecological restoration projects as well as exporting entered data to external geographic information systems;
- querying an underlying database to meet their needs for tracking specific restoration projects;

- reporting and summarizing monitoring data about restoration projects by numerous parameters (e.g., jurisdiction, implementation year, restoration type, planting stock type); and
- implementing adaptive management of ecological restoration practices based on an ever-expanding base of knowledge about the factors that contribute to successful ecological restoration projects.

The STS was operational for all of the Southern Ontario landscape in November of 2007. It can be seen on an Adobe Captivate Video prepared by the Ministry of Natural Resources [11].

COMAP and UWCSG, as a result of the STS, built a mapping interface and database (WIDE image server, with W3C and Open Geospatial Consortium services: WMS, WFS, VML, and SVG) and can serve thematic and highly resolute airphoto data for Southern Ontario. To ensure that all of Ontario is served and easily searchable STS uses GeoBase satellite data and a Lambert projection. STS achieved the following technical objectives:

- a common mapping canvas with airphoto, satellite and thematic data to the extent available in Ontario;
- an ability to enter spatial (polygon) and tabular data about stewardship and restoration projects;
- a stewardship tracking protocol, data model and web service;
- an ability to export/import spatial data to/from external GIS; and
- an ability to query, report and summarize the database to track success of specific projects.

The STS is acknowledged to be the leading effort in shared information infrastructure for cross-scalar provincial reporting on stewardship in Canada. WIDE mapping supports the capture of fine-scale information using standard protocols, which can then be used to augment existing GIS data to permit landscape GIS analyses. WIDE services can increase the value of information management resources by integrating numerous data sources in geographic space in order to provide enhanced direction for asset management decisions. These processes can be manual or automated.

The dynamic asset base, which can be modified and new projects added over time, consists of

- maps;
- a database of projects;
- attribute data related to ecological restoration projects including geographic shapes, text, pictures, audio and video; and
- users, funding agencies and their respective roles.

5.2 CI Based on Dynamic Asset-Mapping for Geographic Communities - Family Service Toronto

Community participants will take the time to organize and share community perspectives, reporting and analyses using collaborative geomatics to populate

maps with text, data and media, if the members of the community see value returning to their community through their active participation. This process known as "dynamic community asset mapping" is an extension of "community asset mapping" as initially defined by John McKnight and John Kretzmann [5] and now widely practiced by communities in North America.

Family Service Toronto (FST), a city-wide social service agency that offers family counseling and community development services, and COMAP have recently formed a partnership to build a portal to support the FST Building Inclusive Communities Division, Community and Neighbourhood Development Unit (CND). CND is funded to facilitate a community development planning project in an area of Toronto that includes established social service agencies, grassroots groups, businesses, faith groups, residents and other interested parties.

The goal of this Community Development Project is to create and implement a community planning process, which is fully inclusive and rooted in best practices of community development and empowerment. The main objective is to increase the amount of community planning that is done collaboratively, inclusively and intentionally. FST believes that one of the best tools to assist in this process is the development of an "open" and accessible web-based asset-mapping process that puts mapping tools into the hands of the groups who have the least resources in the community.

The evolution of grassroots groups from (horizontal) community circles to coherent organizations with capacity to collaborate with (vertically oriented) external resources is seen, by professional Toronto community developers, as an evolution to viable community governance. Community asset mapping in the pursuit of improving assets and capabilities through collaboration, falls short of viability in situations where governance is weak. In the absence of a coherent system of governance, access by grass-root groups to external resources, by default, falls under the control of the external organizations (that do not necessarily reflect the involved input of community residents). The objective here is to develop capable and effective innovative neighbourhood collaboratives to which resources could be devolved by external agencies.

In order to create a forum for collaboration and dynamic asset mapping, we are building the newsatlas^{TM} portal with FST as system custodian. We use the newspaper-map metaphor as a mechanism that would encourage maintenance of current community information. newsatlas^{TM} will pilot community development work in four Toronto neighbourhoods in conjunction with service organizations that participate in FST. The newsatlas^{TM} service architecture will have three components:

1. a public view with organized news pages, map-layer based search facilities, calendars, classifieds and a service directory;
2. a secure social network service for participants who develop and publish newsatlas^{TM} content; and
3. an underlying database that holds content and application services.

newsatlas^{TM} will start as a community asset mapping initiative and be maintained as a community news source with departments and sections providing:

local news, entertainment, arts, sports and recreation content, lifestyle and spiritual content with mapping and event calendars. The process is available to community groups and social service agencies at no charge which levels the playing field for groups with little or no resources. At the outset we envision a Toronto wide service with a list of neighbourhood "front" pages, which will mimic a city-wide newspaper. All content will be searchable city-wide by drawn map area in combination with powerful search tools that support searches by content, space and time.

The newsatlas^{TM} media services will contribute to building viable community governance. newsatlas^{TM} is intended to bring on-going service sustainability in terms of content, community participation and social enterprise revenues, all to address the main project objective: to increase the amount of community planning that is done collaboratively, inclusively and intentionally.

The asset base just described consists of:

- maps;
- databases of community and neighbourhood assets ranging from service agencies to gardens to sports facilities to events; and
- the content supported by newsatlas^{TM}.

The asset base is dynamic and supports CI as various community members can contribute to the databases and the content represented by the newsatlas^{TM} publications.

5.3 Dynamic Asset-Mapping for Communities of Practice the Mennonite Heritage Portrait

The Mennonite Heritage Portrait (MHP) [12] is a social network for a community of practice that contributes digital artifacts and narratives to a rich archival asset base and web presentation of the Mennonite story [13]. The narratives are WebDocs with Wiki links to digital artifacts and other content types such as map location, groups, users, and forum threads. The MHP supports CI through social network approaches to build a learning network and richer content. The resulting system allows a community of interest to contribute, discuss, narrate and authenticate specialized content. To view and use all services, join the MHP social network [12].

The MHP presents and inter-relates current heritage digital media and document collections (such as a photo-negative collection by Peter Entril Snyder a Canadian painter of Mennonite background), in a comprehensive portrait and narrative. Collections like the negative collection have been digitized and presented along with linked narratives to provide context. The MHP connects this and other extensive collections housed in various locations throughout Waterloo Region, and provides collaborative tools for:

- development of narratives and learning materials that link to collection content;

- content searches that use combinations of simple map, tags, text phrase and meta-data parameters to reveal, map and list content by themes and quality of provenance; and
- tools that enable social networks of formal and ad-hoc communities of practice to contribute, map and link content.

Individual heritage groups and heritage sites often operate in information silos, focusing on the content for which they are responsible. A number of museums and collections have begun to digitize their content and make it available online, without reference to material outside their collections. Consequently, many artifacts, media and documents are being displayed in online isolation, without attributions or associations to other collections or material on the same subject. MHP objectives and goals are to:

- present content and narratives previously unavailable online in a coherent, easy-to-navigate and useful manner;
- connect content from various collections seamlessly and present the content in new and meaningful ways, within its proper context;
- engage Mennonite youth with their culture and history through a social network participatory medium;
- support social network services to enable local conversations among practitioners, youth, and the community on cultural heritage topics of common interest; and
- provide a mechanism for grassroots generated content to be authenticated and moved into the authoritative collection, i.e. "the canon."

The MHP provides tools for the development of community narratives with links to a rich database of digital material. Further development of this system to include genealogy and built heritage systems will extend its relevance beyond the immediate community as an attractor for tourism and economic development.

A creative social network service has been created (My Profiles, My Contacts, My Groups, My Messages, My Recommended Content, My Groups with Forums, managed threads and posts, My Bookmarks, Document development with content upload, writing, and WIKI linking) for the Mennonite Heritage Portrait (MHP). This social network, as well as mapping, will be a common feature of most application services in the collaborative innovation framework. Mapping searches also serve to connect individuals where site related activity is of common interest. This platform supports extension of social innovation and best practices beyond the location or community of original work.

6 Related Work

The asset-mapping CI approach proposed in this paper contrasts with closed community efforts, which involve limited interaction, and both restricted knowledge of community assets and their value chains [14]. In the context of service systems, it has been suggested that open service-oriented models could use novel paradigms based on innovation [15] and asset mapping.

Thinking frameworks have been proposed to help organizations focus their management attention, and enable users to participate in the innovation process [16]. In addition, modeling approaches have been defined to describe innovation networks from a services system perspective and to address inter-organizational interactions [10]. In comparison, our proposal focuses on a framework that supports the construction of web-based collaborative innovation systems based on asset mapping that can address both inter-organizational and general community interactions and resource exploitation and production.

In summary, the research described in this paper based on our experiences aims at defining asset-mapping driven approaches to the development of web-based intelligent collaborative innovation systems that promote explicit knowledge sharing needs among cohesive and global-scope communities.

7 Conclusion

This paper outlines the concepts of collaborative innovation based on asset mapping, its importance to the functioning of modern society and how modern web-based tools could be used to support this activity. Based on the experience gained in designing and building over 50 systems that incorporate collaborative innovation, some of which are described in some detail in this paper, it is clear that collaborative innovation takes many forms. Thus, it is not possible to build a single set of tools to support collaborative innovation. Rather a set of meta-tools is needed which can be used to build tailored systems to fit specific situations that arise when web-based collaboration is to occur. The WIDE technologies are an example of the type of meta-tools that are needed. Lessons learned from the construction of WIDE are being applied in the creation of WIDE 2.0.

Acknowledgment

The authors would like to thank the Ontario Ministry of Natural Resources (OMNR), Land Information Ontario, the Ontario Federation of Anglers and Hunters, GeoConnections, the Oak Ridges Moraine Foundation, Family Service Toronto (FST), Canadian Heritage, the Indigenous Cooperative on the Environment (ICE), the Northern Ontario Mushkegowuk First Nations and Mushkegowuk Tribal Council, the Mississaugas of the New Credit First Nation for their financial support and cooperation in implementing the projects described in this paper.

References

1. Vinge, V.: 2020 Computing: The Creativity Machine. Nature 440, 411 (2006)
2. WEF: World Economic Forum Annual Meeting 2008: The Power of Collaborative Innovation, http://www.weforum.org/en/events/ArchivedEvents/AnnualMeeting2008/index.htm

3. Chesbrough, H., Vanhaverbeke, W., West, J.: Open Innovation: Researching a New Paradigm. Oxford University Press, Oxford (2006)
4. Wenger, E.: Communities of Practice: Learning, Meaning and Identity. Cambridge University Press, Cambridge (1998)
5. Kretzmann, J.P.: Building Communities from the Inside Out. ACTA Publications, Skolkie Illinois (1993)
6. Alencar, P., Oliveira, T., Cowan, D., Mulholland, D.: Towards Monitored Data Consistency and Business Processing based on Declarative Software Agents. In: Garcia, A.F., de Lucena, C.J.P., Zambonelli, F., Omicini, A., Castro, J. (eds.) Software Engineering for Large-Scale Multi-Agent Systems. LNCS, vol. 2603, pp. 267–284. Springer, Heidelberg (2003)
7. Bertino, E., Martino, L., Paci, F., Squicciarini, A.: Security for Web Services and Service-Oriented Architectures. Springer, Heidelberg (2010)
8. Longstaff, J., Lockyer, M., Nicholas, J.: The Tees Confidentiality Model: An Authorization Model for Identities and Roles. In: Proceedings of the Eighth ACM Symposium on Access Control Models and Technologies (2003)
9. Masum, H., Tovey, M.: Given Enough Minds . . . Bridging the Ingenuity Gap. Bridging the Ingenuity Gap. First Monday 11(7) (July 2006), http://firstmonday.org/htbin/cgiwrap/bin/ojs/index.php/fm/article/view/1370/1289
10. Janner, T., Schroth, C., Schmidt, B.: Modelling Service Systems for Collaborative Innovation in the Enterprise Software Industry: The St. Gallen Media Reference Model Applied. In: IEEE International Conference on Services Computing, pp. 145–152. IEEE Computer Society, Los Alamitos (2008)
11. STS: STS Video, http://www.comap.ca/STSVid/Prt1Introduction.htm
12. MHP: Mennonite Heritage Portrait, http://www.mennoniteheritageportrait.ca
13. GAMEO: Global Anabaptist Mennonite Encyclopedia Online, http://www.gameo.org
14. Chesbrough, H.: Open Innovation: The New Imperative for Creating and Profiting from Technology. Harvard Busienss School, Boston (2003)
15. Maglio, S., Srinivasan, S., Kreulen, J., Spohrer, J.: Service Systems, Service Scientists, SSME, and Innovation. Communications of the ACM 49(7), 81–85 (2006)
16. van der Walt, J.S., Buitendag, A.A., Zaaiman, J.J., van Vuuren, J.J.: Community Living Lab as a Collaborative Innovation Environment. Issues in Informing Science and Information Technology 6, 421–436 (2009)

A Distributed Dynamics for WebGraph Decontamination

Vanessa C.F. Gonçalves[1], Priscila M.V. Lima[2],
Nelson Maculan[1], and Felipe M.G. França[1]

[1] Systems Engineering and Computer Science Program, COPPE,
Universidade Federal do Rio de Janeiro, Rio de Janeiro, RJ, Brazil
{vcarlaufrj,maculan,felipe}@cos.ufrj.br
[2] Department of Mathematics, Instituto de Ciências Exatas,
Universidade Federal Rural do Rio de Janeiro, Seropédica, RJ, Brazil
priscilamvl@ufrrj.br

Abstract. In order to increase the visibility of a target pageT, web spammers create hyperlink structures called *web bubbles*, or *link farms*. As countermeasure, special mobile agents, called *web marshals*, are deployed in the detection and disassembling of link farms. Interestingly, the process of minimizing the number of web marshals and the number of hops needed to dismantle a web bubble is analogous to the graph decontamination problem. A novel distributed algorithm for graph decontamination, which can be used to define the behavior of web marshals, is introduced in this work. The new algorithm is asynchronous and topology independent. Moreover, it presents equal or better performance and needs smaller numbers of web marshals when compared to recent related works targeting only circulant graphs, a typical structure of link farms.

Keywords: graph decontamination, link farm, distributed algorithms, scheduling by edge reversal, web bubble, web graph.

1 Introduction

Web spammers create a set of web pages, called *web bubble* or *link farm*, having each of its pages pointing to a target web page *T* and to a subset of other pages of the link farm, in order to increase the visibility of page *T*. A number of methods to hide web bubble links have been conceived to work cooperatively with other mechanisms designed to avoid the breakage of these link structures [9][13]. Specialized web crawlers, called *web marshals*, are employed in the detection of this type of "infection" and many of them have been tested in order to improve their efficiency under such attacks [12][15][17].

A number of methods to dismantle link farms can be found in the literature [7][8][11][16]. Considering a web graph containing web bubbles inside its structure as a "contaminated" graph, *graph decontamination*, a graph search problem, has been acknowledged in the literature as the "cure". A web page is contaminated if it includes links to other web pages, forming of a link farm, and all of such pages having also a link to the target page *T*, all of such links inserted by web spammers.

T. Margaria and B. Steffen (Eds.): ISoLA 2010, Part I, LNCS 6415, pp. 462–472.

Among other situations in which the concept of contamination could be identified with, such as a virus spreading throughout hosts, or an exploration team moving through a forest, this work limits itself to the problem of increasing the visibility of a target web page *T* by means of link farms. In such case, decontamination consists in breaking the links that maintain a link farm and not allowing that a page becomes contaminated again. The idea is that, once an unprotected node is exposed to a contaminated neighbor, it can be contaminated again. In the strictest case, the number of contaminated neighbors that can contaminate an unguarded node is equal to one (1). Nevertheless, an acceptable generalization consists in consider that, in the case of web bubbles, a broken link can be restored just by obtaining a simple majority of contaminated neighbors [7].

Special mobile agents, called *web marshals* (WMs)in [7], have been developed in order to perform the decontamination of web pages. They move, from node to node (or page to page), tackling the contamination they are designed to deal with. Luccio and Pagli [7], assuming that circulant graphs are a typical structure used by web spammers, have proposed a distributed algorithm to be embedded into WMs in order to destroy link farms. Both synchronous and asynchronous versions were developed and bounds for the number of WMs and for the number of hops were provided.

This work extends recent results presented in [5] and introduces a new distributed decontamination algorithm based on the *Scheduling by Edge Reversal* (SER) graph dynamics[1][3][4], a distributed algorithm for resource sharing that works in both asynchronous and synchronous modes. The synchronous version of SER works in the following way: starting from any acyclic orientation over the edges of any arbitrarily connected target graph *G*, only sink nodes, i.e., nodes having all of their adjacent edges directed to themselves, are allowed to "operate" upon shared resources. After operating, all sink nodes reverse the orientation of all of their adjacent (incident) edges, becoming source nodes. As a new acyclic orientation having a new set of sink nodes is necessarily defined, the dynamics is indefinitely preserved. The asynchronous version of SER differs only from the synchronous one by allowing each sink node taking any arbitrary time to become a source node. A large spectrum of problems had been tackled with SER, such as: (i) design of asynchronous (clockless) digital circuits [6]; (ii) integrated scheduling of Job Shop and AGVs (Automated Guided Vehicles) in flexible manufacturing systems [18][19]; (iii) biologically plausible rhythm generators [20][21], and (iv) design of collision free MAC protocols [22].

The web graph decontamination based on SER, after an initial acyclic orientation is set, works as follows: each sink node receives a WM; once decontamination is done at a sink node, new WMs are sent, via replication, just to immediate neighbor nodes that will become sinks next, upon edge reversal. The SER-based approach implicitly associate the amount of concurrency provided by a SER dynamics to the number of concurrently operating WMs and the total number of decontamination steps performed, i.e., one hop is counted each time a node receives one (or more) WM copy (or copies) in order to become a new sink. It will be shown that the SER-based approach produces the same or better quality solutions found in [7], while still able to work in arbitrary connected topologies, mostly targeting web graphs [10], and under more strict contamination rules, what suggests its appropriateness on dealing with new kinds of web attacks.

2 Related Works

A distributed algorithm to destroy a link farm F with the help of WMs was proposed by Luccio and Pagli [7]. Equivalent to the mobile agents used in [8][11], which are passed through the links that form the link farm, once a WM gets into a contaminated page the link to the target page T is destroyed.

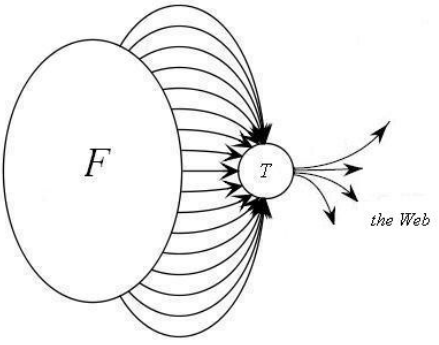


Fig. 1. The structure of a link farm F: increasing the visibility of web page T

Assuming that a link farm F has the structure of a circulant graph (see Figure 1), their work is strongly based on the properties of this particular graph topology. It is demonstrated that in a circulant graph $C_{i,n}(L)$, with L being a list of integers, $k = |L|$, $k + 2$ WMs are needed to dismantle F. Figure 2 illustrates circulant graphs having (a) $k = 3$ (L being {1, 2, 4}), and (b) $k = 2$ (L being {1, 2}).

Based on the visibility of each node (distance in hops from this node to any other node in the graph), another graph decontamination method is proposed in [11]. It is shown that visibility 2, i.e., when an WM can "see" the node hosting it, its immediate neighbors and the neighbors of its immediate neighbors, is enough to avoid overuse of WMs. The total number of employed WMs is one of the main concerns of the present work.

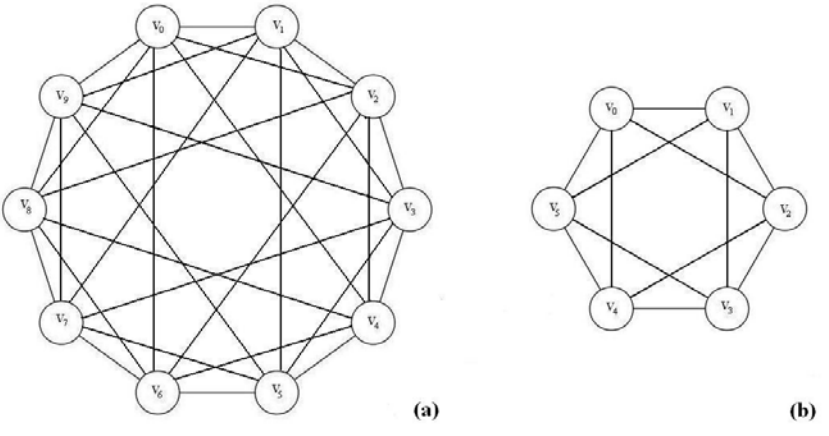


Fig. 2. Examples of circulant graphs: (a) $C_{i,10}(1, 2, 4)$; (b) $C_{i,6}(1, 2)$

3 Edge Reversal Decontamination

It is possible to start a decontamination dynamics based on the SER behavior from any acyclic orientation ω of the target web graph G. It is worth noticing that there is always a node coloring of G associated to ω in the following way: each node receives color equal to the length of the longest directed path from it to a sink node. This is called the *sink decomposition* of ω, where sink nodes receive color $\lambda = 0$. By placing Web Marshals (WMs) into sink nodes, it is easy to see that nodes having color $\lambda = 1$ are next to turn into color $\lambda = 0$ (sinks) upon edge reversal.

As minimizing the number of WMs is a main concern, it is important to observe that having sink decompositions of large length increase the probability of dealing with fewer concurrent sink nodes, i.e., a smaller number of WMs in the web graph. The next two subsections presents (i) the *Alg-Stretcher*, a heuristics to enlarge the length of sink decompositions of already defined acyclic orientations produced by *Alg-Edges* [2], a randomized distributed algorithm designed to produce acyclic orientations over anonymous networks, and; (ii) the *Alg-Decontamination* algorithm, the edge reversal based distributed algorithm; applied, in this work, to circulant graphs used to build link farms. Having in mind the particular properties of circulant graphs, this work also aims to compare the performance, both in terms of the number of WMs and the number of steps taken to decontaminate, between the algorithm proposed by [7] and the SER-based strategy introduced here.

3.1 Alg-Stretcher

Let $\lambda = l_{max}$ be the outer layer in the sink decomposition of a target acyclic orientation ω. *Alg-Stretcher* works as follows: from layers $\lambda = (l_{max} - 1)$ to $\lambda = 0$ in the sink decomposition, each node v in a layer $\lambda = l_v$, $0 \leq l_v < l_{max}$, is tested about being moved to a new outer layer $\lambda > l_v$. If an increase in the number of layers of the sink decomposition of the graph is obtained, the resulting acyclic orientation ω' is kept by having all of v's edges oriented according to the direction of the sink decomposition. If there is no increase in the sink decomposition, the previous orientation is kept.

Lemma 1. *Alg-Stretcher does not induce directed cycles in G.*

Proof. Let ω be an acyclic orientation generated by *Alg-Edges*, for example. By definition, any sink decomposition of ω has no directed path from an inner layer ($\lambda < l_v$) to an outer layer ($\lambda > l_v$). The *Alg-Stretcher* procedure reverses only the orientation of edges incidents to a node v, belonging to layer $\lambda = l_v$, that were coming from nodes of outer layers ($\lambda > l_v$) that became oriented in the opposite direction of the sink decomposition. Figure 3 illustrates the sink decomposition of ω, from layers $\lambda = l_{max}$ to $\lambda = 0$.

In Figure 3 one notes that the node $\boldsymbol{x}$ is part of the path of $\boldsymbol{z}$ and $\boldsymbol{y}$ to sink nodes. If one tests $\boldsymbol{x}$ to move to an outer layer, one has the orientation ω' illustrated by Figure 4. Reversing only the edges from outer layers that would be in the opposite direction of the sink decomposition, one prevents the creation of a cycle in the path to the sink nodes from the elected node v in the sink decomposition. In the previous orientation ω, $\boldsymbol{z}$ and $\boldsymbol{y}$ have no directed paths from them to themselves as well as $\boldsymbol{x}$. So the only

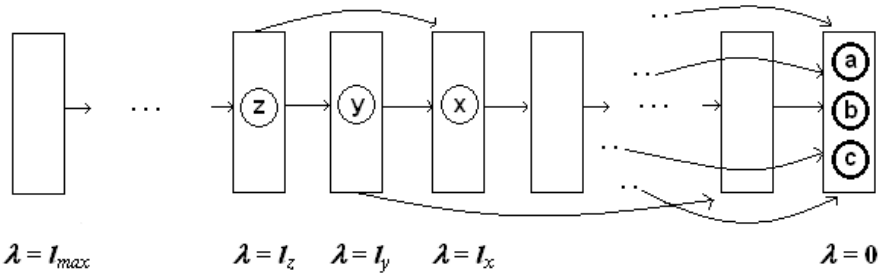


Fig. 3. Sink decomposition of ω, $l_z > l_y > l_x$

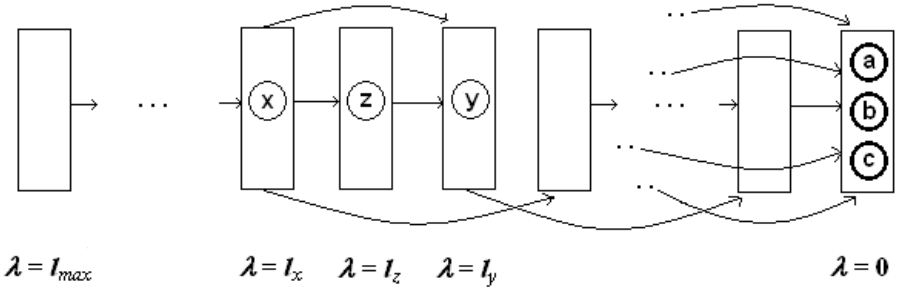


Fig. 4. Sink decomposition of ω, $l_x > l_z > l_y$

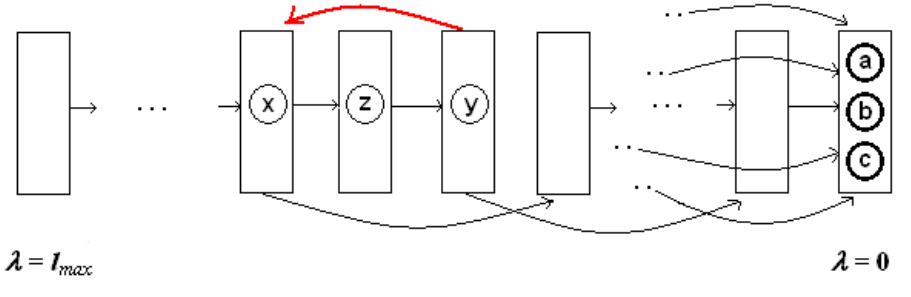


Fig. 5. The creation of a directed cycle

way that one could create a cycle would be a change of only one edge directed from z or y to x. Figure 5 demonstrate that by not reversing the orientation of the edge {y,x}, a cycle is created. Since all the edges that could create a cycle from such nodes are reversed, i.e., keeping the sink decomposition direction, the graph acyclicity is preserved. ■

Figure 6 depicts experimental results from the application of *Alg-Stretcher* in acyclic orientations produced by *Alg-Edges*. Each point is given by the mean value of 500 runs over connected random graphs having the same density. Compared to other two

distributed heuristics studied in [2][23], *Alg-Edges* is the algorithm producing the largest number of colors and *Alg-Stretcher* has induced a further expressive increase in the resulting number of colors.

One expects that the total number of WMs needed for decontamination, to be placed at sink nodes, will be near minimum. It is also worth noticing that to find an acyclic orientation associated to (i) the minimum number of colors [1][3][4], and to (ii) the maximum number of colors [23], are both NP-complete problems.

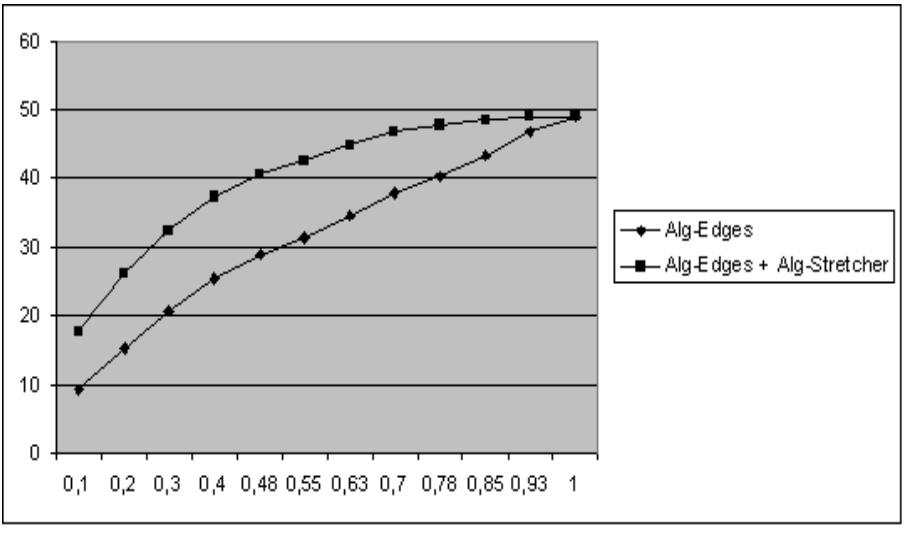


Fig. 6. Applying *Alg-Stretcher* over *Alg-Edges*: numbers of colors vs. graph density

3.2 Alg-Decontamination

Consider that each node of a target directed acyclic graph G can be in the following three local states:

- **Contaminated:** the node is harmful;
- **Clean:** the node is not contaminated;
- **Guarded:** the node contains a WM.

The following is how *Alg-Decontamination* works. WMs are placed into sink nodes ($\lambda = 0$); sink nodes are **guarded** and all other nodes are considered **contaminated** (nodes have visibility 1, i.e., a WM can visualize only nodes hosting it and its immediate neighbors).

Each node checks its own λ while it stills **contaminated**. If the node is **contaminated** and $\lambda = 0$, then it checks if it has received only one WM. If not, i.e., more than one WM has been received, the node chooses just one of them. Then, the chosen WM cleans the node, destroy the links of the web bubble and to the target page *T,* and then makes a decision: (i) terminate, by declaring the node **clean**, and move to other node (a node in $\lambda = 1$), or; (ii) keep the execution, i.e., the node is **guarded**, and make copies of itself and send to the neighbor(s) that will become

sink(s), i.e., neighbors in $\lambda = 1$. A WM cannot terminate its execution unless the majority of its neighbors are **clean**.

Upon sending WM copies, a WM send also a message reversing all incident edges to all of its immediate neighboring nodes, thus producing a new acyclic orientation on G. Nodes that were in $\lambda = 1$ get into $\lambda = 0$, receiving one or more WM copies. At this point, the process repeats itself until all nodes in the graph are **clean**.

According to *Alg-Decontamination*, if a node is a sink ($\lambda = 0$) then it has received at least one WM. When a WM finishes its operation, by reversing all of its directed edges, such node is no longer a sink ($\lambda > 0$). By proceeding with edge reversal over an acyclic orientation, the next orientation will be also acyclic. So, all others nodes that were previously in $\lambda > 0$ will eventually become sinks, guaranteeing the decontamination of all nodes. Notice that by keeping a WM into a node having any immediate neighbor with the majority of its neighbors **contaminated**, we avoid that an already **clean** node becomes a contaminated node once again, as shown in [7].

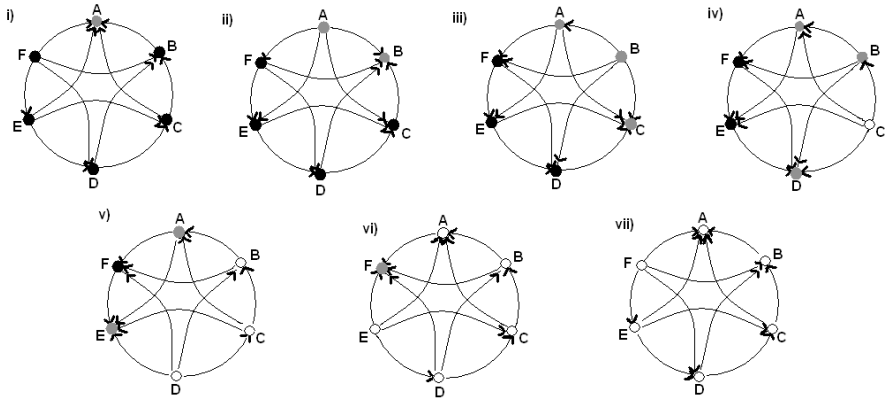


Fig. 7. SER-based decontamination of $C_{i,6}$ (1, 2)

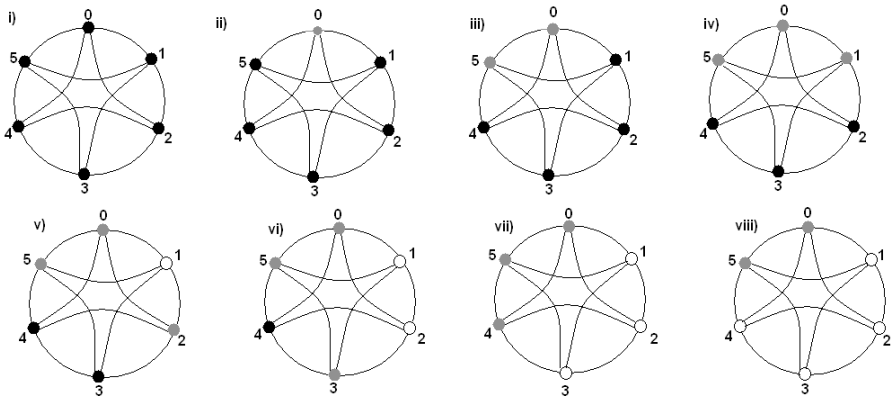


Fig. 8. $C_{i,6}$ (1, 2) decontamination proposed in [7]

The SER-based decontamination of a circulant graph $C_{i,6}$ (1, 2) is illustrated in Figure 7. The behavior of the asynchronous algorithm proposed in [7] running over the same circulant graph $C_{i,6}$ (1, 2) is shown in Figure 8. In both Figures 7 and Figure 8, **contaminated**, **guarded** and **clean** nodes are shown in black, gray, and white colors, respectively.

Theorem 1. *Alg-Decontamination decontaminates any connected graph G = (N, E).*

Proof. Consider an arbitrary connected graph *G=(N, E)* and an acyclic orientation $\omega_0$over all edges in *E*. A property of SER [1][3][4] is that all nodes in *N* will eventually become sink nodes, at least once, in a sequence found inside a finite set of acyclic orientations $\{\omega_0, \omega_1, \omega_2, \ldots, \omega_n\}$, where $\omega_0 \neq \omega_1 \neq \ldots \neq \omega_{n-1} \neq \omega_n$. Necessarily, a period of length *p*, given by the subset of acyclic orientations $\{\omega_i, \ldots, \omega_n\}$, $0 \leq i < n$, $p = n - i + 1$, will be reached. Notice that, in Figure 7, the period is given by the orientations in $\{\omega_3, \omega_{10}, \ldots, \omega_{vi}\}$, and the orientation ω_{vii} is equal to ω_3. Also, notice that all nodes become sinks and a node that is sink in ω_t is no longer a sink in ω_{t+1}, such that a sink in ω_{t+1} has at least one neighbor that is a sink in ω_t[1]. In *Alg-Decontamination*, at the first time a node becomes a sink, it receives a copy of aWM and, since all nodes eventually become sinks, they all end up receiving at least one WM.

Another concern is the contamination *criterium*, i.e., how a **clean** node can get contaminated, which is defined by the number of **contaminated** neighbors of a **clean** node. In the more strict case, this number is equal to one (1) and the necessary condition to keep a node clean would be having all of its neighbors either **clean** or **guarded**. In *Alg-Decontamination*, re-contamination is prevented by local decisions about keeping WMs inside nodes that not reach the necessary condition that avoids get **contaminated** by neighbors. ■

4 Experimental Results

A quantitative comparison between the asynchronous algorithm proposed in [7] and *Alg-Decontamination*, both algorithms were applied over the same set of graph instances, is provided here. The initial acyclic orientation chosen as starting point for *Alg-Decontamination* reproduces the initial scenario used by the algorithm proposed in [7], in which WMs move only in the links of the main cycle.

Circulant graphs $C_{i,n}(L)$ with $k = 2$ ($L = \{1,2\}$) and $k = 3$ ($L = \{1,2,3\}$) are considered. In the case of $k = 2$, *Alg-Decontamination* needed three WMs, while the algorithm proposed in [7] needed four WMs. In the case of $k = 3$, *Alg-Decontamination* needed four WMs, while the proposed in [7] needed five WMs (all tests made with $10 \leq n \leq 10{,}000$).

It is assumed in [7] that the link hops performed by WMs can be counted as the time that decontamination takes to terminate. This way, a comparison is made based on the number of link hops that are needed to decontaminate a graph. The number of

hops needed in [7] is $n - c + h$, where n is the number of nodes, $c = \lfloor (k + 1)/2 \rfloor$ and h is the number of link hops needed to place the first WMs, given by:

$$h = \begin{cases} 3(k^2/4 - k/2) & \text{for } k \text{ even} \\ 3(k^2/4 - k/2 + 1/4) & \text{for } k \text{ odd} \end{cases} \quad (1)$$

About the number of hops, for$k = 2$, *Alg-Decontamination* and the [7] algorithm both needed the same number of hops, which for *Alg-Decontamination* is constant with k, and always takes $n - 1$ hops in circulant graphs. From $k = 3$ upwards, *Alg-Decontamination* needs less link hops to decontaminate, as illustrated by Table 1. This happens since *Alg-Decontamination* doesn't need to perform the hops taken in the algorithm in [7] by a verifier WM about the other WMs' positions.

Table 1. Link hops (k=3)

	Alg-Decontamination	*[Luccio and Pagli 2007]*
10	9	11
50	49	51
1000	999	1001
3000	2999	3001
5000	4999	5001
10000	9999	10001

5 Conclusions

Alg-Decontamination is a new distributed algorithm for the decontamination of web graphs. Compared to recent related work [7], *Alg-Decontamination* provided the same or better figures in terms of the number of web marshals and number of hops taken. Moreover, while the said related work is dedicated to the class of circulant graphs, *Alg-Decontamination* is topology independent. Although a major concern of this work is about providing a topology independent distributed algorithm, i.e., a SER-Based decontamination able to deal with any given graph structure, that could not be demonstrated and exercised in this paper. This suggests that *Alg-Decontamination* could be used in new/unseen forms of web spam. A heuristic to obtain acyclic orientations associated to the maximum number of node colors, i.e., large sink decompositions, was also produced. Devising a combinatorial optimization approach to this problem is left for future work.

Acknowledgments. The authors would like to acknowledge the Web Science Brasil project, CNPq 557.128/2009-9 and FAPERJ E-26/170028/2008 (Programa INC&T - Projeto: Instituto Brasileiro de Pesquisa em Ciência da Web).

References

1. Barbosa, V.C., Gafni, E.: Concurrency in Heavily Loaded Neighborhood-Constrained Systems. ACM Transactions on Programming Languages and Systems 11(4), 562–584 (1989)
2. Arantes Jr., G.M., França, F.M.G., Martinhon, C.A.: Randomized generation of acyclic orientations upon anonymous distributed systems. Journal of Parallel and Distributed Computing 69, 239–246 (2009)
3. Barbosa, V.C.: An Introduction to Distributed Algorithms. The MIT Press, Cambridge (1996)
4. Barbosa, V.C.: An Atlas of Edge-Reversal Dynamics. Chapman & Hall/CRC, London (2000)
5. Gonçalves, V.C.F., França, F.M.G., Maculan, N., Lima, P.M.V.: SER-Based Web Graph Decontamination. In: 1st Worskhop INCT WebScience, PUC-RIO (2010)
6. Cassia, R.F., Alves, V.C., Bernard, F.G.-D., França, F.M.G.: Synchronous-to-asynchronous conversion of cryptographic circuits. Journal of Circuits, Systems, and Computers 18, 271–282 (2009)
7. Luccio, F., Pagli, L.: Web Marshals Fighting Curly Link Farm. In: Crescenzi, P., Prencipe, G., Pucci, G. (eds.) FUN 2007. LNCS, vol. 4475, pp. 240–248. Springer, Heidelberg (2007)
8. Flocchini, P., Nayak, A., Schulz, A.: Cleaning an arbitrary network with mobile agents. In: Chakraborty, G. (ed.) ICDCIT 2005. LNCS, vol. 3816, pp. 132–142. Springer, Heidelberg (2005)
9. Gyöngyi, Z., Garcia-Molina, H.: Web Spam Taxonomy. In: Proc. of AIRWeb 2005, Chiba (2005)
10. Donato, D., Leonardi, S., Millozzi, S., Tsaparas, P.: Mining the inner structure of the Web graph. In: 8th International Workshop on the Web and Databases (WebDB 2005), Baltimore, Maryland (2005)
11. Flocchini, P., Nayak, A., Schulz, A.: Decontamination of Arbitrary Networks using a Team of Mobile Agents with Limited Visibility. In: Proc. of 6th IEEE/ACIS International Conference on Computer and Information Science, ICIS 2007 (2007)
12. Becchetti, L., Castillo, C., Donato, D., Leonardi, S., BaezaYates, R.: Link-Based Characterization and Detection of Web Spam. In: Proc. AIRWeb 2006, Seattle (2006)
13. Du, Y., Shi, Y., Zhao, X.: Using Spam Farm to Boost Page Rank (2006) (manuscript under publication)
14. Lapaugh, A.: Recontamination does not help to search a graph. Journal of the ACM 40(2), 224–245 (1993)
15. Barrière, L., Flocchini, P., Fraignaud, P., Santoro, N.: Capture of an intruder by mobile agents. In: Proc. 14th ACM Symposium on Parallel Algorithms and Architectures (SPAA), Winnipeg, Canada (2002)
16. Luccio, F., Pagli, L., Santoro, N.: Network Decontamination with Local Immunization. In: Proc. of the 8th APDCM, pp. 110–118 (2006)
17. Borie, R., Tovey, C., Koenig, S.: Algorithms and Complexity Results for Pursuit-Evasion Problem. In: Proc. of the 21st International Joint Conference on Artificial Intelligence, pp. 59–66. Morgan Kaufmann Publishers, San Francisco (2008)
18. Lengerke, O., Carvalho, D., Lima, P.M.V., Dutra, M.S., Mora-Camino, F., França, F.M.G.: Controle distribuído de sistemas job shop usando escalonamento por reversão de arestas. In: Proc. of the XIV Latin Ibero-American Congress on Operations Research (CLAIO 2008), Cartagena de Indias (2008) (in Portuguese)

19. Lengerke, O., Dutra, M.S., França, F.M.G., Tavera, M.J.M.: Automated Guided Vehicles (AGV): Searching a Path in the Flexible Manufacturing Systems. Journal of Konbin 8, 113–124 (2008)
20. Yang, Z., França, F.M.G.: A generalised locomotion CPG architecture based on oscillatory building blocks. Biological Cybernetics 89(1), 34–42 (2003)
21. Braga, R.R., Yang, Z., França, F.M.G.: Implementing an Artificial Centipede CPG: Integrating appendicular and axial movements of the scolopendramorph centipede. In: Proc. of the International Conference on Bio-inspired Systems and Signal Processing (BIOSIGNALS 2008), vol. 2, pp. 58–62. INSTICC Press (2008)
22. Pinho, A.C., Santos, A.A., Figueiredo, D.R., França, F.M.G.: Two ID-Free Distributed Distance-2 Edge Coloring Algorithms for WSNs. In: Fratta, L., Schulzrinne, H., Takahashi, Y., Spaniol, O. (eds.) IFIP-TC 6. LNCS, vol. 5550, pp. 919–930. Springer, Heidelberg (2009)
23. Arantes Jr., G. M.: Tracks, Concurrency Optimization, and Probabilistic Initialization on Systems under Edge Reversal, D.Sc. thesis, COPPE/UFRJ (2006) (in Portuguese)

Increasing Users' Trust on Personal Assistance Software Using a Domain-Neutral High-Level User Model

Ingrid Nunes, Simone D.J. Barbosa, and Carlos J.P. de Lucena

PUC-Rio, Computer Science Department, LES - Rio de Janeiro, Brazil
{ionunes,simone,lucena}@inf.puc-rio.br

Abstract. People delegate tasks only if they trust the one that is going to execute them, who can be a person or a system. Current approaches mostly focus on creating methods (elicitation approaches or learning algorithms) that aim at increasing the accuracy of (internal) user models. However, the existence of a chance of a method giving a wrong answer decreases users' trust on software systems, thus preventing the task delegation. We aim at increasing users' trust on personal assistance software based on agents by exposing a high-level user model to users, which brings two main advantages: (i) users are able to understand and verify how the system is modeling them (transparency); and (ii) it empowers users to control and make adjustments on their agents. This paper focuses on describing a domain-neutral user metamodel, which allows instantiating high-level user models with configurations and preferences. In addition, we present a two-level software architecture that supports the development of systems with high-level user models and a mechanism that keeps this model consistent with the underlying implementation.

1 Introduction

As web applications become increasingly interactive, accessible, and pervasive the web is providing mechanisms that can help users extend their mental and physical capabilities. The Web now provides access to huge amounts of well-organized information and supports social interactions well beyond our physical limitations. Thus there are new challenges in managing both the quantity of information and the complexity and timeliness of relationships. Multi-agent Systems (MASs) [1], with roots not only in Artificial Intelligence (AI) but also in distributed systems and software engineering, can incorporate autonomous behavior to support web users in meeting many of these new barriers by freeing users from repetitive and tedious tasks. In addition, MASs, by providing autonomous behavior, may be employed by web users to support access to information and decision-making. The concept of user (or personal) agents was championed by Maes in 1994. In [2], she introduced the idea that autonomous agents may be personal assistants who are collaborating with the user in the same work environment. However, even though significant research effort has been invested on developing user agents, we are far from their massive adoption.

T. Margaria and B. Steffen (Eds.): ISoLA 2010, Part I, LNCS 6415, pp. 473–487, 2010.

People delegate tasks only if they trust the one that is going to execute them, who can be a person or a system. For user agents, this claim is supported by the study presented by Schiaffino & Amandi in [3]. Their study showed that users fear having a completely automated agent. In addition, they concluded that a large group of users is willing to adopt user agents only if they know exactly what the agent is going to do, i.e. if they trust that the agent will perform a behavior previously approved by them. Current approaches mostly focus on creating methods (elicitation approaches or learning algorithms) that aim at increasing the accuracy of (internal) user models. Nevertheless, the risk that a method gives a wrong answer decreases users' trust on software systems, thus preventing the task delegation.

Our research aims at increasing the level of acceptance of user agents (and task delegation to computer systems) by users. In order to achieve this our goal is to increase users' trust on personal assistance software based on two main properties: (i) transparency; and (ii) power of control. The main idea is to expose user models (or profiles) in an end-user-readable manner, and therefore users can understand the system behavior (transparency). In addition, users can manage their model to control the agents' behavior (power of control).

In this paper we demonstrate steps in creating a model-driven approach for developing personal assistance software based on high-level user models and user agents. We present an approach to empower users with a high-level domain-specific language that allows them to dynamically program and personalize their agents. Even though inference models might reach the wrong conclusions about user preferences and cause agents to take inappropriate actions, they can be leveraged to create initial versions of the user model, so that users can make fine-grained modification on it. The steps to be presented are: (i) a high-level user metamodel to represent user configurations and preferences; and (ii) a software architecture to build personal assistance software based on agents that are adapted driven by instances of the proposed metamodel.

The proposed Domain-specific Model (DSM) (high-level user metamodel) provides the necessary vocabulary to build an end-user configurations and preferences language. Existing representation models of user preferences force users to express their preferences in a particular way. Consequently, these works create the need for elicitation techniques to interpret answers to questions and indirectly build the user model. The language that our DSM creates allows users to configure their agents and to express different kinds of preference statements, creating a vocabulary that allows representing statements close to the ones expressed in natural language. The proposed DSM is an application-domain-neutral metamodel that may be instantiated to build different applications.

The paper also describes a software architecture to build personal assistance software that follows our approach. The architecture is composed of user agents that are dynamically adapted based on a user model that follows our metamodel. In this sense, users' customizations are represented in a high-level user model and realized at the implementation-level by user agents. As configurations and preferences may impact in different agents and their components, the user model

becomes a modularized view of users' customizations, facilitating their management, considering that they change over time. However, given that there are users' customizations represented in two different levels of abstractions, we also present an algorithm that keeps both representations consistent.

This paper is organized as follows. Section 2 describes our metamodel. In Section 3, we describe the software architecture for building personal assistance software. Section 4 presents an evaluation of our metamodel by showing its generality when used across different domains. Section 5 presents related work. Finally, Section 6 concludes this paper.

2 A Domain-Neutral User Metamodel

One of the most challenging tasks in building personal assistance software able to act on users' behalf is to capture particularities of the person being represented (user model) so that the system can present an appropriate behavior. We aim at exposing the user model to users in order to provide them the power of controlling their agents and increasing the trust on the system. This user model must be expressed with very high-level abstractions, otherwise users are not able to understand them. This section presents our proposal of a domain-neutral high-level user metamodel to represent users' customizations.

We acknowledge the relevance of the existence of a reasoning algorithm for user models. However, our goal is to use the proposed metamodel in a level higher than the ones processed by algorithms. When users inform their preferences to applications with restricted user models, e.g. models with boolean preferences, they have to translate preferences statements expressed in natural language to the one imposed by the application. Our goal with this high-level model is to provide users with a larger vocabulary to express their preferences and leave the task of translating preferences for a particular model to the system, as Figure 1 illustrates. For instance, a user may say: "My preference is not to buy a laptop of the brand X, but I don't care if the brand is A, B or C." Suppose now that the system uses a reasoning algorithm to process preferences expressed as a partial order relation. We can represent that preference like this:

$$preference = \{< A, X >, < B, X >, < C, X >\}.$$

Therefore, our metamodel is implementation-independent.

It is important to highlight that our user metamodel distinguishes user *configurations* from *preferences*, which we collectively refer to as *customizations*. Configurations are direct and determinant interventions that users perform in a system, such as adding/removing services or enabling optional features. They can be related to environment restrictions, e.g. a device configuration. They are represented by optional and alternative features that users choose for customizing their application. A feature, in turn, is any variable characteristic of the system. On the other hand, preferences represent information about the users' values that influence their decision making, and thus can be used as resources in agent reasoning processes. They typically indicate how user rates certain options

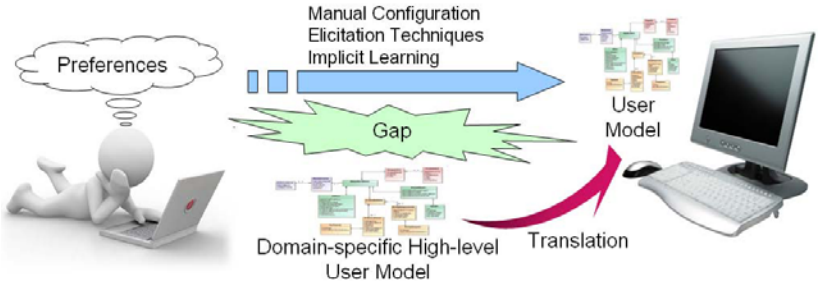


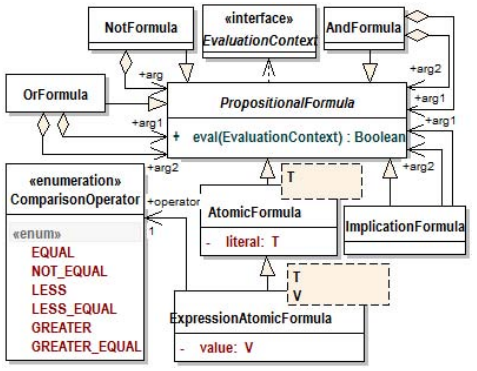
Fig. 1. High-level User Model

better than others in certain contexts. They are a representation of the cognitive model of the user in order for agents to behave and make decisions in a way as close as possible as users would do.

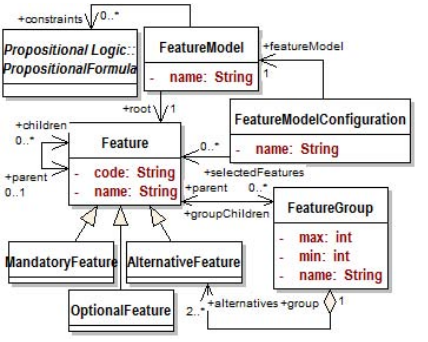
Our user metamodel is instantiated in a stepwise fashion. First, application developers instantiate part of the metamodel for defining application-specific abstractions and constraints. This is performed at development time. Second, at runtime, the user instantiates preferences and configurations in order to customize the personal assistant application. Our metamodel, which is an extension of the UML metamodel[1], is depicted in Figure 2 in four different parts. Elements of the UML metamodel, e.g. `Class` and `Property`, are either distinguished with a gray color in diagrams or are referred to in properties. Next we describe our metamodel more extensively.

There are three models that must be instantiated at development time: (i) Ontology model; (ii) Feature model; and (iii) Preferences Definition model. The Ontology model represents the set of concepts within the domain and the relationships between those concepts. The Feature model (Figure 2(b)), in turn, allows modeling variable traits within the domain, which are later used for defining user configurations. This model incorporates the ideas of Software Product Lines (SPLs) [4] and their feature models [5]. SPL is a new software reuse approach that aims at systematically deriving families of applications based on a reusable infrastructure with the intention of achieving both reduced costs and reduced time-to-market. The goal of the Feature model is to describe variation points and variants in the system, which can be either optional or alternative, and can be added and removed dynamically from the application. A `FeatureModel` is a tree of `Features`. A `Feature` can be mandatory, optional and alternative. Mandatory features are represented only if they are in a chain with other optional and alternative features, and therefore need to be represented. Otherwise, they will be present in the system any way. `AlternativeFeatures` are grouped into `FeatureGroups`, which define the minimum and maximum number of alternatives that can be chosen. Finally, constraints may be defined in order to represent relationships between variations, e.g. a feature requires the presence

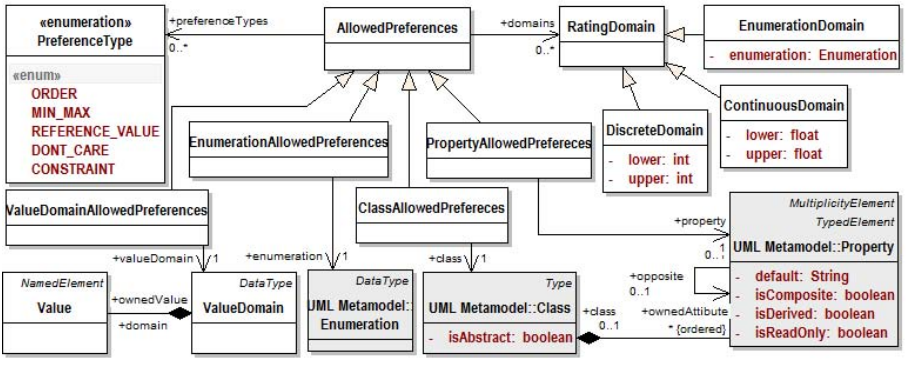
[1] http://www.omg.org/spec/UML/



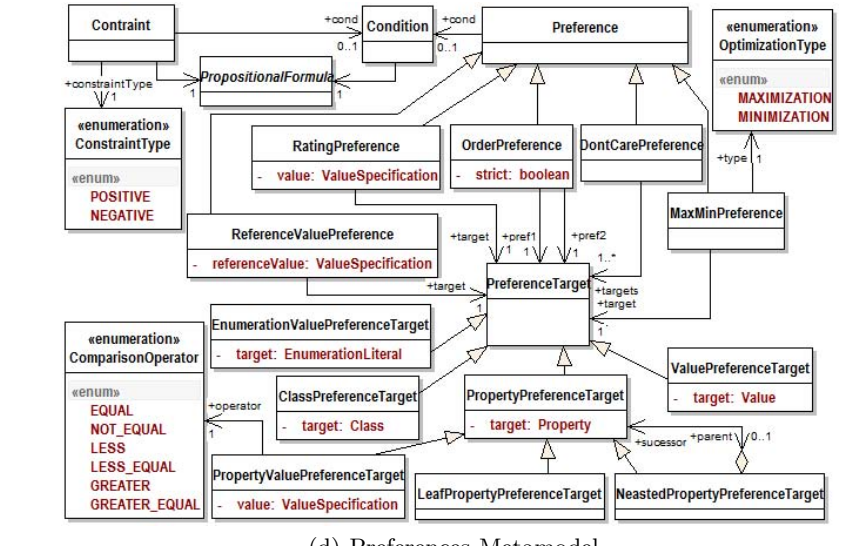
(a) Propositional Logic



(b) Feature Model



(c) Preferences Definition Metamodel



(d) Preferences Metamodel

Fig. 2. User Metamodel

of another. These constraints are expressed as `PropositionalFormulas`, shown in Figure 2(a).

The goal of our metamodel is to allow users not only to customize their applications with selected features, as in a SPL, but also to capture users' preferences in order for automated customized agents to act appropriately on their behalf. The Preferences Definition model defines restrictions over preferences that users can express. Its metamodel is presented in Figure 2(c). The purpose of this model is to define *how* users can express their preferences and *about which elements* of the Ontology Model. Even though it is desirable that users be able to express preferences in different ways, it is necessary to have agents that can deal with them. For instance, if application agents can deal only with quantitative preference statements, user preferences expressed in a qualitative way will have no effect on the system behavior if there is no mechanism to translate them to quantitative statements.

Users can express different types of preference: (i) Order (`ORDER`) – expresses an order relation between two elements, allowing users to express *"I prefer trains to airplanes."* A set of instances of the Order preference comprises a partial order; (ii) Reference Value (`REFERENCE_VALUE`) – enables users to indicate one or more preferred values for an element. It can be interpreted as the user preference is a value on the order of the provided value; (iii) Minimize/Maximaze (`MIN_MAX`) – indicates that the user preference is to minimize or maximize a certain element; (iv) Don't Care (`DONT_CARE`) – it allows indicating a set of elements that users do not care about. It is useful for users to express *"I don't care if I travel with company A, B or C;"* (v) Rating – allows users rating an element. By defining a `RatingDomain` for an element, users can rate this element with a value that belongs to the specified domain. This domain can be numeric (either continuous or discrete), with specified upper and lower bounds. In addition, an enumeration can be specified, e.g. LOVE, LIKE, INDIFFERENT, DISLIKE and HATE. Moreover, different domains can be specified for the same element. Using Rating preferences, it is possible to assign utility values to elements, or to express preference statements; and (vi) Constraint (`CONSTRAINT`) – a particular preference type that establishes a hard constraint over decisions, as opposed to the other preference types, used to specify soft constraints. Constraints allows users to express strong statements, e.g. *"I don't travel with company D."*

Different kinds of preferences may be used by agents in different ways, according to the approaches they are using to reason about preferences. If an agent uses utility functions and the user defines that the storage capacity of a computer must be maximized and provides a reference value α, the agent may choose a utility function like $f(x) = \sqrt[\alpha]{x}$.

For defining the allowed preference types, developers must create instances of `AllowedPreferences`, and make the corresponding associations with types and domains. The specializations of `AllowedPreferences` characterize different element types that can be used in preference statements. There are four different possibilities: classes (*I prefer notebook to desktop*), properties (*The notebook weight is an essential characteristic for me*) and their values (*I don't*

like notebooks whose color is pink), enumeration literals (*I prefer red to blue*) and values (*Cost is more relevant than quality*). Value is a first-class abstraction that we use to model high-level user preferences. Values are essential when using a value-focused thinking [6]: *"Values are what we care about. As such, values should be the driving force for our decision making. They should be the basis for the time and effort we spend thinking about decisions."* A scenario that illustrates the use of values is in the travel domain. A user may have comfort (a value) as a preference when choosing a transportation, instead of specifying fine-grained preferences, such as *trains are preferred to airplanes*, but *traveling in an airplane first-class is better than by train*, and so on. In this case, the user agent is a domain expert that knows what comfort means.

Based on these three instantiated models and on our Preferences metamodel (Figure 2(d)), it is possible to build a User Model to model preferences and configurations. It is composed of two parts: (i) Configuration model; and (ii) Preferences model. As discussed above, in the Configuration model, users choose optional and alternative features (variation points) from the Feature model, defining their configurations, which are instance of the `FeatureModelConfiguration`. Therefore, a configuration is a valid set of selected optional and alternative features of a `FeatureModel`. On the other hand, in the Preferences model, users define a set of preferences and a set of constraints. These are more closely related to a cognitive model of the user. User preferences (or soft constraints) determine what the user prefers, and indirectly how the system *should* behave. If the preferred behavior is not possible, the system may move to other acceptable alternatives. Constraints, in turn, are restrictions (hard constraints) over elements. As opposed to preferences, they directly define mandatory or forbidden choices that *must* be respected by the system.

Figure 2(d) shows the `Constraint` element and five different specializations of `Preference` that represent the different preference types previously introduced. Constraints are expressed in propositional logic formulae, however using only $\neg$, $\wedge$ and $\vee$ logical operators. Atomic formulae refer to the same types of elements of preferences and can use comparison operators ($=$, $\neq$, $>$, $\geq$, $<$, $\leq$) between properties and their values. The `PreferenceTarget` and its subtypes are used to specify the element that is the target of the preference statement or formula. In addition, it allows to specify nested properties, such us `Flight.arrivalAirport.location.country`. If we have directly associated preferences to classes, properties, enumerations and values, either we would have to make specializations of each preference type to each element type or to change the UML metamodel to make a common superclass of classes, properties, enumerations and values. Given that we did not want to modify the UML metamodel, but only to extend it, and the first solution would generate four specializations for each preference type, we used the `PreferenceTarget` as an indirection for elements that are referred in preferences and constraints.

Besides defining preferences and constraints, users can specify conditions, also expressed in propositional logic formulae (Figure 2(a)), to define contexts in which preferences and constraints hold. Furthermore, in order to guarantee that

users produce valid instances of the metamodel, we have defined additional constraints over instantiated models, e.g. in a nested property, the child of a property whose class is X must also be a property of Class X.

3 A Two-Level Software Architecture for Building Personal Assistance Software

The main contribution of this paper is the user metamodel described previously. However, we also present a software architecture that provides a structure of modules and incorporates the idea of a high-level user model. This software architecture addresses the domain of personal assistance software. The structure of this architecture aims at building systems composed of personalized user agents which are adapted based on user models, which can be instantiated and modified by end-users.

The goal of the proposed architecture is to accommodate a high-level user model, but also to allow building high quality personal assistance software by taking into account good software engineering practices. User customizations may be seen as a *concern* in a system that is spread all over the code. However, at the same time, each customization is associated with different services (also concerns) provided to users. Therefore, when developing such system one has to choose the dimension in which the software architecture will be modularized: in terms of services (Figure 3(a)) or modularizing user settings in a single model (Figure 3(b)). It can be seen that it is not possible in either approach to modularize concerns in single modules. In addition, without modularizing user customizations, as in Figure 3(a), they are buried inside the code, thus making it difficult to understand them as a whole.

Moreover, user preferences play different roles in agent architectures [7,8]. We illustrate examples of these roles in Table 1. If all this information is contained in a single user model, we have the problems discussed above and this model would aggregate information related to different concerns of the system (low cohesion among user model elements).

Our solution is to provide a *virtual separation of concerns* [9]. A concern is anything that is interesting from the point of view of a stakeholder. In our case, the concern that will be virtually modularized is the user model. The main idea is to structure the user agent architecture in terms of services by modularizing its variability as much as possible into agent abstractions. We provide a virtual

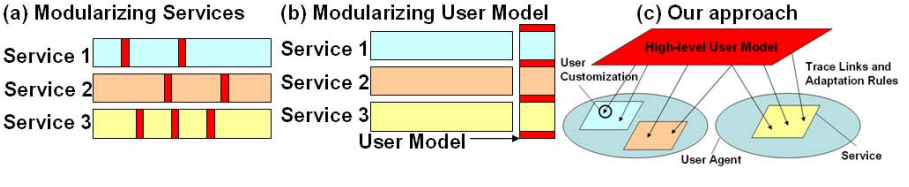


Fig. 3. Modularization Approaches

Table 1. User Preferences and their roles into agent architectures

Attitude	**Example**
Goal	I want to drink red wine.
Belief	I like red wine.
Motivation	Red wine is good for the heart.
Plan	In order to drink red wine either I go to the supermarket and buy a bottle (plan A) or I go to my friend's home who always have wine there (plan B).
Meta-goal	I want to drink red wine, but spending less money as possible (so I might choose plan B).

modularized view of user customizations, as Figure 3(c) illustrates. Customizations are not design abstractions, but they are implemented by typical agent abstractions (goals, plans, etc.), i.e. they play their specific roles in the agent architecture. The virtual user model is a complementary view that provides a global view of user customizations. This model uses a high-level end-user language, and users are able to configure their agents by means of this model. Using a high-level user model to drive adaptations on personal assistance software brings the main following advantages: (i) user customizations are implementation-independent; (ii) the vocabulary used in the user model becomes a common language for users specifying configurations and preference; (iii) the user model modularizes customizations, allowing a modular reasoning about them.

3.1 Detailing our Software Architecture

In this section we detail our proposed architecture, depicted in Figure 4, and describe the mechanism that makes the high-level user model (henceforth referred to as user model) work with agent architectures.

The *User Agents* module consists of agents that provide different services for users, e.g. scheduling and trip planning. Their architecture supports variability related to different users, and provides mechanisms to reason about preferences.

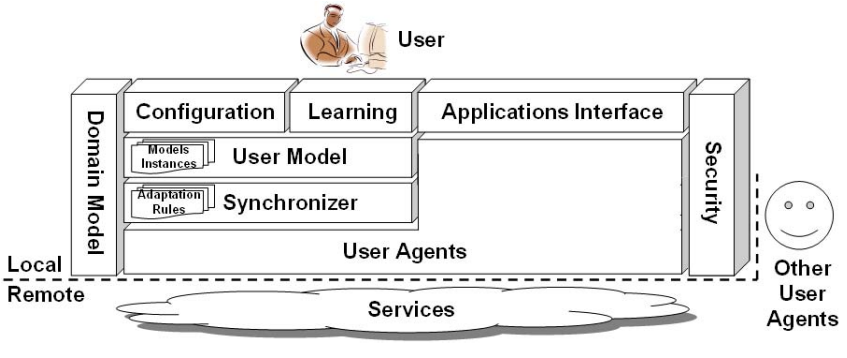


Fig. 4. Proposed Architecture

We propose to adopt an agent-based approach to design and implement user-customizable systems for several reasons: (i) agent-based architectures are composed of human-inspired components, such as goals, beliefs and motivations, thus reducing the gap between the user model (problem space) and the solution space; (ii) plenty of agent-based AI techniques have been proposed to reason about user preferences, and they can be leveraged to build personalized user agents; and (iii) agent architectures are very flexible, thus facilitating the implementation of user customizations. For instance, there is an explicit separation between what to do (goals) and how to do it (plans).

User Agents use services provided by a distributed environment (the *Services* cloud), and their knowledge is based on the *Domain Model*, composed of entities shared by user agents and services, application-specific, etc. The *Security* module addresses security and privacy issues, because user agents may share information with other user agents. This module aggregates policies that restrict this communication, assuring that confidential information is kept safely secured. Users access services provided by user agents through the *Applications Interface* module.

The *User Model* contains user configurations and preferences expressed in a high-level language. They are present in the user agents architecture but as design-level abstractions. By means of the *Configuration* module, users can directly manipulate the *User Model*, which gives them the power to control and dynamically modify user agents, using a high-level language. In addition, changes in the *User Model* may be performed or suggested by the *Learning* module, which monitors user actions to infer possible changes in the *User Model*. This module has a degree of autonomy parameter, so it may automatically change the *User Model*, or just suggest changes to it, to be approved by the end users.

The *User Model* and *User Agents* are connected by representing user customization in two different levels of abstraction. This connection is stored in the form of trace links, indicating how and where a customization is implemented in a user agent(s). Adaptations are performed at runtime and are accomplished based on the trace links between the *User Model* and the *User Agents* architecture. The *Synchronizer* is the module in charge of adapting *User Agents* based on changes in the *User Model*. It is able to understand these trace links, and it knows which transformation must be performed in the *User Agents* based on changes in the *User Model*. Therefore, the *User Model* drives adaptations in the *User Agents*.

The algorithm we define to be performed by the *Synchronizer* module is presented in Algorithm 1. It receives as input a previous and an updated versions of a user model, as well as a map containing rules that states which set of actions must be performed when an event (a change) on the user model occurs. The algorithm first calculates the set of events (changes) between the two versions of the user model (line 1). Next, for each event, it gets the set of rules that "listen" to the event (line 2–6). The result is the set of rules that produce actions for the event set. Then, it is retrieved, from each rule, the set of actions (changes) that must be performed at the implementation level of the system (lines 7–9), so that it turns to be consistent with the updated version of the user model. Finally, all actions are performed (lines 10–11).

Algorithm 1. Adaptation algorithm

Input: UM: previous user model; UM': updated user model; $rulesMap$: adaptation rules mapped to the events they observe

```
1  events = diff(UM, UM');
2  adaptationRules = ∅;
3  foreach Event e ∈ events do
4      rules = e.get(rulesMap);
5      if rules ≠ null then
6          adaptationRules = adaptationRules ∪ rules;
7  actions = ∅;
8  foreach Rule r ∈ adaptationRules do
9      actions = actions∪ r.getAdaptationActions(UM, UM', events);
10 foreach Action a ∈ actions do
11     a.doAction();
```

Examples of actions are the addition or removal of agents, beliefs, goals and plans. Events are the (de)selection of features or addition or removal of a preference over a certain element. And rules associate events with actions.

4 Instantiating Our User Metamodel for Different Application Domains

Our metamodel was built using preference statements collected from different individuals in a user study and from papers related to user preferences. The idea was to contemplate the different kinds of preference statements in order to maximize the users' expressiveness. The metamodel uses abstractions from the user preferences domain, therefore the language is built as an end-user language. This section presents two Preferences models to show that our metamodel is generic enough to model different kinds of preferences statements in different domains – flight reservation and computer purchase domains. Given that these are two

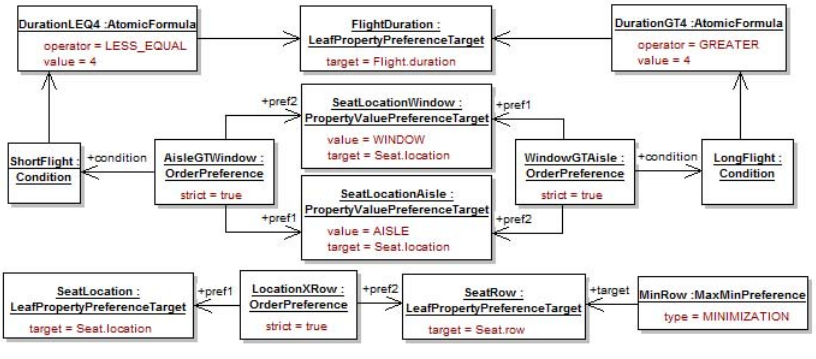


Fig. 5. User Preferences model in Flight Domain

well-known domains, we assume that the reader is familiar with them, and due to space restrictions, we present only the Preferences models. In addition, we assume that the Preferences Definition model defines that all preference types over all elements are allowed.

The first Preference model, which is from the flight domain, indicates where a user prefers to sit inside an airplane. This model consists of three order preferences, two of them with conditions, and one minimization preference. Next, we present the four modeled preference statements in natural language, and Figure 5 shows how they are modeled with our metamodel abstractions.

P1. *If the flight is short, i.e. its duration does not exceed 4 hours, I prefer a seat by the aisle to a seat by the window.*
P2. *If the flight is long, i.e. its duration is higher than 4 hours, I prefer a seat by the window to a seat by the aisle.*
P3. *I always prefer to sit at the first rows of the airplane.*
P4. *Sitting at the first rows of the airplane is more important to me than the seat location.* The computer domain Preferences model presented in Figure 6 has some elements in gray color. They are not part of the Preferences model, but from the Domain model, but we included them in Figure 6 to present some application-specific concepts used in this model. First, four values are defined in the Computer Domain (mobility, readability, performance and cost). These values can be rated with "+", ranging from one to five. These are the natural language preference statements modeled in Figure 6:

P1. *Cost is the most important value (+++++).*
P2. *I rate performance with ++++.*
P3. *I rate readability with ++++.*
P4. *I rate mobility with ++.*
P5. *I'm expecting to pay around $800 for my laptop.*
P6. *I want a computer with less than 3Kg.*
P7. *The lighter the computer is, the better.*

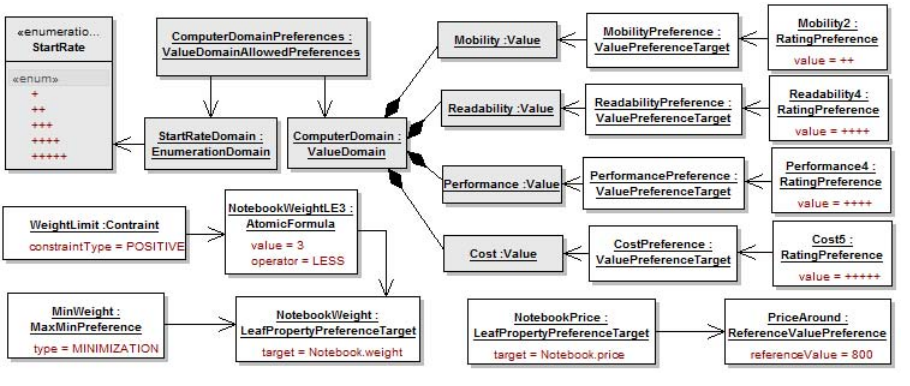


Fig. 6. User Preferences model in Computer Domain

It is important to notice that Rating and Order preferences provide different information. By saying that cost is +++++ and performance is ++++, a user is informing that cost is more important than performance (order), but performance is also important, and should be taken into account.

5 Related Work

Several approaches have been proposed to deal with user preferences. To build our metamodel, we have conducted extensive research on which kinds of preferences other proposals represent and additional concepts they define. Typically, preferences are classified as quantitative or qualitative (e.g. "I love summer" versus "I like winter more than summer"). Both approaches can be represented through our metamodel. Quantitative preferences are modeled in the framework proposed in [10] by means of a preference function that maps records to a score from 0 to 1. On the other hand, CP-Nets [11] models qualitative preferences. CP-Nets also allow modeling conditionality, which is considered in our work as well. The concept of normality is defined in [12], so that users can express preferences considering normal states of the world, but these preferences may change when the world changes. The normality abstraction can be modeled using conditions in our metamodel.

Ayres & Furtado proposed the OWLPref [13], a declarative and domain-independent preference representation in OWL. This work has the same purpose of our work in the sense that it generically models user preferences. However, OWLPref does not precisely define the preferences model, e.g. lacking the definition of associations, it shows only a hierarchical structure of preferences. A preference metamodel is also proposed in [14]. However, its expressiveness is very limited. It only allows to define desired values (or intervals) of object properties.

One of the biggest projects in the context of personalized user agents is the Cognitive Assistant that Learns and Organizes (CALO) project[2] [15,16], whose goal is to support a busy knowledge worker in dealing with the twin problems of information and task overload. Along the project, the research effort was mostly concentrated in the PTIME agent, which is an autonomous entity that works with its user, other PTIME agents, and other users, to schedule meetings and commitments in its user's calendar. Users are able to express their preferences, nevertheless the adopted language is tight to application domain (meeting scheduling). Despite this limitation, the CALO project substantially advanced on the development of user agents, also taking into account human-computer interaction (HCI) issues that are essential for improving the chances of users adopting personal agents. Therefore, lessons learned from this project [16] can be leveraged in our work.

6 Conclusion

With the growth of the Internet, interactivity and access to information are significantly increasing. At the same time, several of our everyday-tasks are being

[2] http://caloproject.sri.com/

managed by software applications, such as to-do lists and schedules. The combination of these trends converge to the automation of user tasks performed by agents that act on behalf of users. Agents must have reliable user models that assure they act appropriately, otherwise they will not be trusted by users.

In order to increase users' trust on personal assistance software based on automated agents, we proposed in this paper the idea of exposing high-level user models to users so that they can verify and understand this model as well as control it by configuring it and making fine-grained modifications. Our proposal is a domain-specific metamodel that provides abstractions from the user domain, including configurations, constraints and preferences. Different abstractions used by end users in natural language statements are directly represented. Users are able to tailor personal assistance software systems with optional and alternative features, and model their preferences. Besides (hard-)constraints, five different preferences types (soft-constraints) can be represented: order, rating, reference value, maximization/minimization and don't care. In addition, we adopt values as a first-class abstraction to model high-level preferences. Instances of our metamodel are to be used in combination with our proposed software architecture, which uses them as a global view of user customizations. Services are provided by user agents structured with traditional agent-based architectures. The User Model provides a modularized view of different user-related concepts spread into agent architectures. We also presented an algorithm to be performed by the Synchronizer module, which ensures that changes in the User Model demands appropriate adaptations in user agents.

We are currently working in several directions. First, we made a survey of preference statements provided by user in order to build a language based on our metamodel using syntactic sugar. In addition, we are investigating how to verify the User Model to identify inconsistencies across preferences. Finally, we are implementing a framework based on our software architecture to provide a solid infrastructure to build personal assistance software systems.

Acknowledgements

This work has been partially supported by CNPq 557.128/2009-9 and FAPERJ E-26/170028/2008. It is related to the following topics: Software technologies for web applications - A Multi-Agent Systems Approach for Developing Autonomic Web Applications - G1. Design techniques to improve the development of autonomic Web applications. Ingrid Nunes #141278/2009-9, Simone Barbosa #313031/2009-6, Carlos Lucena #304810/2009-6 also thank CNPq for respective research grants.

References

1. Weiss, G. (ed.): Multiagent systems: a modern approach to distributed artificial intelligence. MIT Press, Cambridge (1999)
2. Maes, P.: Agents that reduce work and information overload. Commun. ACM 37(7), 30–40 (1994)

3. Schiaffino, S., Amandi, A.: User - interface agent interaction: personalization issues. Int. J. Hum. Comput. Stud. 60(1), 129–148 (2004)
4. Pohl, K., Böckle, G., van der Linden, F.J.: Software Product Line Engineering: Foundations, Principles and Techniques. Springer, Heidelberg (2005)
5. Kang, K., Cohen, S., Hess, J., Novak, W., Peterson, A.S.: Feature-oriented domain analysis (FODA) feasibility study. Technical Report CMU/SEI-90-TR-021, SEI (1990)
6. Keeney, R.L.: Value-focused thinking – A Path to Creative Decisionmaking. Harvard University Press, London (1944)
7. Doyle, J.: Prospects for preferences. Computational Intelligence 20, 111–136 (2004)
8. Nunes, I., Barbosa, S., Lucena, C.: Modeling user preferences into agent architectures: a survey. Technical Report 25/09, PUC-Rio, Brazil (September 2009)
9. Kästner, C., Apel, S.: Virtual separation of concerns - a second chance for preprocessors. Journal of Object Technology 8(6), 59–78 (2009)
10. Agrawal, R., Wimmers, E.L.: A framework for expressing and combining preferences. In: 2000 ACM SIGMOD, pp. 297–306 (2000)
11. Boutilier, C., Brafman, R.I., Hoos, H.H., Poole, D.: Cp-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. Journal of Artificial Intelligence Research 21, 135–191 (2004)
12. Lang, J., van der Torre, L.: From belief change to preference change. In: ECAI 2008, The Netherlands, pp. 351–355. IOS Press, Amsterdam (2008)
13. Ayres, L., Furtado, V.: Owlpref: Uma representação declarativa de preferências para web semântica. In: XXVII Congresso da SBC, Brazil, pp. 1411–1419 (2007)
14. Tapucu, D., Can, O., Bursa, O., Unalir, M.O.: Metamodeling approach to preference management in the semantic web. In: M-PREF, USA, pp. 116–123 (2008)
15. Berry, P., Peintner, B., Conley, K., Gervasio, M., Uribe, T., Yorke-Smith, N.: Deploying a personalized time management agent. In: AAMAS 2006, pp. 1564–1571 (2006)
16. Berry, P.M., Donneau-Golencer, T., Duong, K., Gervasio, M., Peintner, B., Yorke-Smith, N.: Evaluating user-adaptive systems: Lessons from experiences with a personalized meeting scheduling assistant. In: IAAI 2009, pp. 40–46 (2009)

Understanding IT Organizations

Claudio Bartolini[1], Karin Breitman[2], Simone Diniz Junqueira Barbosa[2], Mathias Salle[1], Rita Berardi[2], Glaucia Melissa Campos[3], and Erik Eidt[1]

[1] Hewlett-Packard
[2] Departamento de Informática, PUC-Rio
[3] Universidade do Estado do Rio Grande do Norte
Rua Marquês de São Vicente, 225 – Gávea – Rio de Janeiro – RJ – Brasil – 22451-900
{claudio.bartolini,mathias.salle,erik.eidt}@hp.com,
{karin,simone,rita}@inf.puc-rio.br

Abstract. Understanding IT organization is essential for ensuring a successful transformation phase in IT outsourcing deals. We present a model of IT organization, a methodology for deriving it – based both on ethnography and data mining - and a suite of tools for representing and visualizing the model, and to help design changes to bring the organization from its current (AS-IS) state to a desired (TO-BE) state, along with tools for comparing models of organizations based on qualities and characteristics that are expected to have a bearing on the success of the IT transformation step.

Keywords: network of responsibilities, knowledge elicitation, contextual design.

1 Introduction

Large IT outsourcing deals often involve a wholesale takeover of large chunks of the outsourcer's IT organization by the outsourcing firm. Even before such deals enter their execution phase, the pursuit organization of the outsourcing firm is faced with the daunting task of making sense of the AS-IS state of the outsourcer's IT organization, including people, processes, systems, applications.

It is true that enterprise directory, policies, process description and other documents can begin to provide a picture of what the organization is formally supposed to look like, but the reality of things can be quite different from that picture.

The current state of things is that the learning process that enables IT transformation is more of an art than a science. That results in slower time to IT transformation, which in turn directly translates to a financial loss to the IT outsourcing company.

Our project on **Understanding IT Organizations** aims at providing a useful, actionable model of the IT organization that the outsourcing firm can use to spot opportunities for simplification, decision support and automation. Our contributions include:

T. Margaria and B. Steffen (Eds.): ISoLA 2010, Part I, LNCS 6415, pp. 488–501, 2010.

1. A simple yet expressive meta-model for IT organizations, able to represent roles, their responsibilities, and the artifacts that embody the information that roles exchange between them in carrying out their work
2. A light-weight methodology for deriving a model for the organization at hand, mining the disparate sources of information available throughout the IT organization, eliciting information from IT staff through ethnographic studies and reconciling where possible inconsistencies and conflicts
3. A visual modeling environment that
    a. Represents the information currently contained in the model showing an AS-IS map of the IT organization.
    b. Allows a user to author a picture of the desired state of the organization (TO-BE).
4. A suite of reasoning tools to help the user spot opportunities for simplification, decision support and automation in the organization. Such tools
    a. Assist design of the TO-BE state of the organization
    b. Highlight necessary changes to take the organization from the AS-IS state to the TO-BE state.
5. A library of archetypical models of IT organizations to assist design of TO-BE states, complete with qualities and characteristics that such designs enables (see Jahn & Lawson [14]).
6. Tools for comparative analysis of AS-IS IT organization against archetypical models.

The benefits that our approach can bring to an outsourcing firm embracing it include being able to drive down cost, increase margin and improve customer satisfaction.

The remainder of this paper is structured as follows. In section 2, we spell out our architectural principles. In section 3, we describe relevant design context and refer back to relevant literature in which some of the concepts were first introduced. In section 4, we derive and describe our meta-model of IT organizations. In section 5, we present our light-weight methodology to populate instances of IT organization models. In section 6, we review related work. Finally, in section 7, we discuss applicability of our meta-model, methodology and tools; we lay out our plans for future work and conclude.

2 Principles

In this section, we lay out the architectural principles that we follow in our work. The architectural principles we follow in our work are the following. First, we assume "**multiple versions of the truth**". When dealing with such complex systems as IT organizations, inconsistency and incompleteness are facts of life. Because the driving factor for us is to shorten the time to IT transformation, we just assume that there will be cases where information is incomplete or inconsistent. Our methodology does prescribe a reconciliation step. However, this is done on a best-effort basis, ensuring a quicker time to results especially when compared and contrasted with methods such as ARIS (quote), which assume a "single version of the truth". In order to realize this principle, in our architecture we borrow the concept of *viewpoints* from requirement engineering.

Second, we "**itís not just what is said that matters, but who says it**". We couple the concept of viewpoint, with thorough analysis of provenance. Provenance is particularly important given that we are not striving for removing all conflicts, but rather allowing for some degree of uncertainty where different information sources disagree (be they people or systems/documents).

Third, "**not all sources are equally credible**". We also aim at establishing and maintaining reputation measures of the various information sources.

Finally, we privilege obtaining broad brush, sketchy pictures of the IT organization. The principle is to go over a **breadth-first, iterative** refinement of the IT organization model so obtained. Again, this is consistent with our aim of faster IT transformation.

3 Relevant Design Concepts

3.1 Contextual Design

Contextual design is a customer-centered design process which uses ethnographic methods to gather extensive field data and to elaborate models for understanding users' needs, tasks, intents, and processes, i.e., for "seeing work". Their ultimate goal is to design products and systems that meet both users' and business needs [2, 12]. The five work models employed in contextual design are [2, p.86]:

- *flow model*: the communication and coordination necessary to do the work. It represents the individuals who do the work, their responsibilities or roles, the groups of people who have common goals or who take action together, the communication between people to get the work done, the communication topics and artifacts passed between people, the places people go in and out of while doing their work, and breakdowns (problems) in communication or coordination;
- *sequence model*: the detailed work steps necessary to achieve an intent. It represents the intent that the sequence is designed to achieve, a trigger that causes the sequence of actions, the steps (actions or thoughts preceding an action), the order, loops, and branches that connect the steps, and breakdowns in executing one or more steps;
- *artifact model*: the physical things created to support the work, along with their structure, usage, and intent. It represents the information presented by the artifact, its parts, structure, and presentation aspects, annotations and conceptual distinctions that reflect different usages of the the artifact, and breakdowns in using it;
- *culture model*: to represent constraints on the work caused by policy, culture, or values. It represents influencers, the kinds (e.g., standards, policy, power, values, identity, emotions, style, and preferences) and extent of their influence on the work, and the breakdowns interfering in the work;
- *physical model*: to show the physical structure of the work environment as it affects the work. It represents the places in which work occurs, the physical structures and movement within the space, the hardware, software, artifacts, and other tools, as well as their layout in the environment, and breakdowns showing how the physical environment interferes in the work.

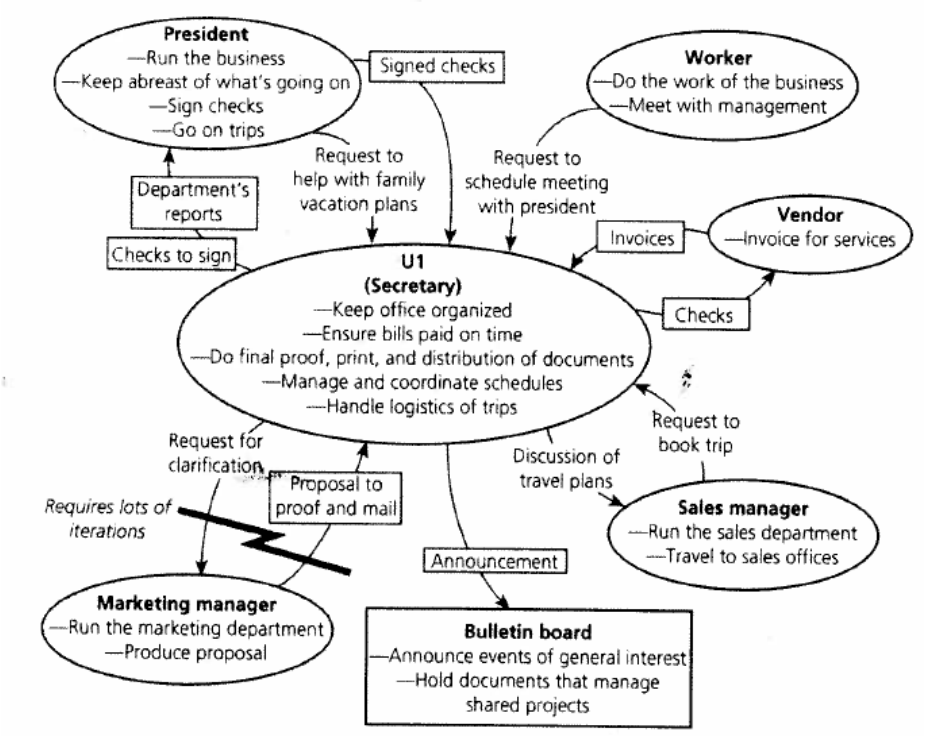


Fig. 1. A sample flow model

Figure 1 depicts a sample flow model of some secretarial work [2].

The ellipses represent roles or invididuals and their responsibilities; the rectangles represent artifacts; the arrows represent the flow of information and artifacts between people; and the lightning bolt represents a breakdown.

One may notice the importance assigned to breakdowns in every model used in contextual design. They represent mainly the need for an intervention and thus have a prominent role in supporting decision-making processes within an organization.

Although useful for understanding work, contextual design presents some important limitations regarding our project. First, it presents one version of the facts as truthful. It does not allow us to represent different views of the organization acquired from different information sources. Consequently, it does not explicitly allow us to represent uncertain data, data with varying degrees of reliability, nor data in different levels of abstraction. It does not help us to trace a piece of information back to its source. In addition, it does not make the distinction between responsibility and accountability (see next section). Finally, it does not provide an underlying model to bind together the information from the various models, which would help us detect inconsistencies, conflicts, and also opportunities for analogical reasoning that may prove useful for learning and reuse of both processes and information.

3.2 Accountability

Responsibility and accountability are terms that are often used interchangeably, but they are not. In particular, responsibility is only one of the aspects defining a role, and it is to be intended as doing the work to achieve a task. Accountability, on the other hand, is a different concept and is best exemplified by President Truman's statement "the buck stops here". Responsibility can be shared, as more than one role may work together in order to complete a task, even though that most often signals that co-ordination is necessary to avoid likely duplication of work, and might indicate that a further refinement step is necessary to break down the task into sub-task for which different roles may be responsible. Accountability instead is never shared, and is therefore a more defining aspect of a role.

In our work, we borrow from the concept of Responsibility Assignment Matrix, or RACI model [3]. In its most widely accepted variant the RACI model defines the *key responsibility roles* as follows:

Responsible
Those who do the work to achieve the task. There is typically one role with a participation type of Responsible, although others can be delegated to assist in the work required (the derived *RASCI* model also includes a separate identification fro those who participate in a supporting role, namely *supporting*).

Accountable
Those who are ultimately accountable for the correct and thorough completion of the deliverable or task, and the one to whom Responsible is accountable. In other words, an Accountable must sign off (Approve) on work that Responsible provides. There must be only one Accountable specified for each task or deliverable.

Consulted
Those whose opinions are sought; and with whom there is two-way communication.

Informed
Those who are kept up-to-date on progress, often only on completion of the task or deliverable; and with whom there is just one-way communication.

Another requirement for our model is then to allow us to clearly represent the key responsibility roles defined in the RACi model.

3.3 Trust and Reputation of Information Sources

According to Hertzum [10], the trust in information sources is central to establish their quality. He argues assessing "the quality of an information source is essentially a matter of establishing to what extent one is willing to place trust in it". Hertzum et al. [11] conducted two studies in which people (individuals, project groups, and organizations) and documents (electronic and paper documents as well as information systems) were experienced as different types of sources, which had to be treated differently.

Contrary to the common belief that engineers follow a principle of least effort by choosing their information sources on the basis of ease of access rather than quality of

contents, Hertzum [10] showed that the engineers prefer sources with a known or easily determinable trustworthiness as much as information that is easily accessible. He found that software engineers devote significantly more attention to quality-related factors than to cost-related factors.

Tseng and Fogg [28] distinguish four types of trust by means of the evidence on which the trust is founded:

- First-hand experience (e.g. interacting with people over time, we assess their expertise and trustworthiness).
- Reputation; that is, what third parties have reported (e.g. asking someone for advice based on having her recommended by a colleague).
- Simple inspection of surface attributes (e.g. assessing people by the way they dress or the language they use).
- General assumptions and stereotypes (e.g. believing that your friends tell the truth, whereas car salespeople do not).

Even a person who is overall trustworthy will most likely have different degrees of knowledge about the diverse subjects she refers to, inasmuch as documental information sources will provide details about certain subjects and abstract details about others. Moreover, people may be aware of and willing to disclose their own lack of knowledge about certain subjects. Therefore, it is important that our model makes it possible to establish not only an absolute assessment of reliability an information source, but also an additional assessment relative to the specific subjects being discussed, and the source's own confidence on their assertions.

3.4 Provenance and Traceability

Provenance and traceability refer to the origin of information and physical artifacts and products. There are several works describing provenance and traceability in diverse contexts (e.g., software engineering, the semantic web, industrial product chains), and even efforts to produce a standard for describing provenance (cf. the W3C incubator group at http://www.w3.org/2005/Incubator/prov/wiki/Requirements).

According to Miles and colleagues [21], data provenance research needs to answer the following questions:

- Which data items generated a particular data product?
- What computations generated these data items?
- Where did the computations occur?

To be able to answer these questions, they enummerate the relations between data items, which correspond to the refinements or transformations that data undergo in the context of a workflow.

In understanding organizations, we are concerned with provenance in at least two different levels: the source of information and artifacts depicting the organization, and the source of information and artifacts used or generated by the organization. Our model needs to make the distinction between these two levels.

3.5 Case-Based Reasoning

Case-based reasoning (CBR) is a reasoning technique based on a base of cases which contain knowledge about specific prior episodes, in context, and mostly at an operational level [18, 15]. The goal is to record lessons learned, and to be able to reuse this knowledge in new circumstances. A case is mainly composed of a problem/situation description, a solution, and an outcome (the state of the world resulting from having applied the solution to the described situation) [15]. A case can represent, among other things: how to achieve one or more goals, the preconditions for achieving them and how to get to the state where the preconditions hold, which problems may arise in achieving a goal, and the effects of an action performed in certain circumstances. New solutions are generated by retrieving similar (and most relevant) cases from the case base and adapting them to fit new situations.

Figure 2 depicts the case-based reasoning cycle [15]. In the retrieval, we assess the situation, elaborate a case description, compute possible indexes for the new situation, search the case base for partially matching cases, and select the best case(s). Having retrieved the best cases, it is necessary to assess how well they fit into the current situation and, if necessary, adapt them by insertion, substitution, transformation or deletion of some elements. The critique and evaluation of the potential solution may lead to additional adaptation and, when a good solution is reached, it may compose a new case to be stored in the case base.

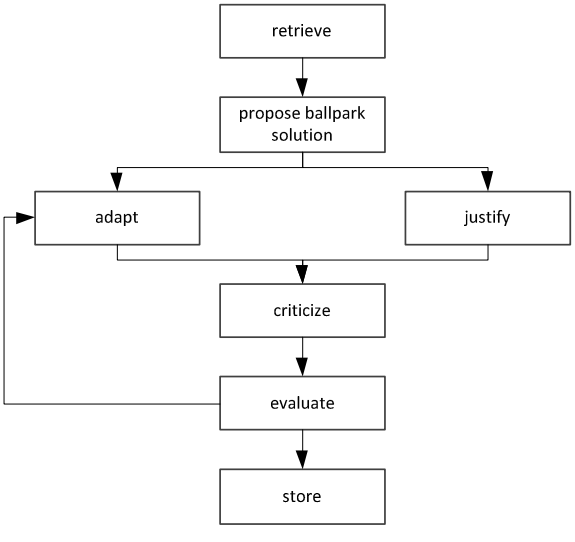


Fig. 2. The case-based reasoning cycle [15].

The retrieve, reuse, revise and retain cycle in CBR is an important resource both for knowledge management and process (re)engineering. One of the major challenges in CBR is to model information and index the cases in such a way that they can be easily searched for, compared, and retrieved. A requirement for our metamodel is to provide enough structure for such indexing and characterization of cases.

4 Proposed Metamodel

Our metamodel basically allows us to represent propositions that answers to the 6W questions (Who, What, When, Where, Why, and hoW - Figure 3).

The metamodel allows us to represent the aforementioned contextual design work models. The flow model is represented by the *product–agent* relations; the sequence model by the *agent–task*, *task–task*, and *task–event* relations; the artifact model indirectly by means of the relations that go from *goal* to *product*, and encapsulating some details within the *artifact* element, using the contextual design model for detailing it when necessary; the culture model by the *agent–agent* relations, and the physical model by the *location–location* and *task–system* relations.

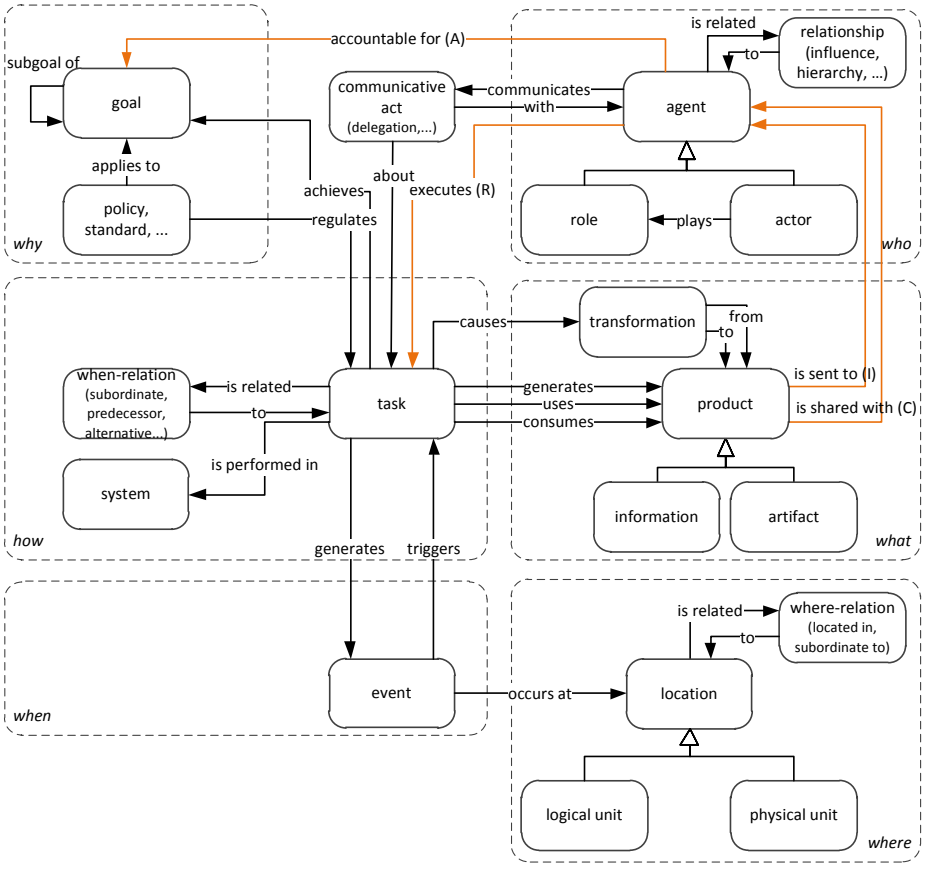


Fig. 3. Proposed metamodel

In Figure 3, we have identified the relations that represent elements of the RACI model, to allow us to represent accountability concerns:

- R: <agent, executes, task>
- A: <agent, is accountable for, goal>

- C: <product, is shared with, agent>
- I: <product, is sent to, agent>

From the metamodel, we extract propositions that depict the organization and relate them both to their information sources and to evaluations of the proposition (Figure 4). Such evaluations are made by people in the organization or inferred by reasoning systems, based on the consistency of the propositions acquired from different information sources and their reputation. This helps us to establish the provenance of the information about the organization.

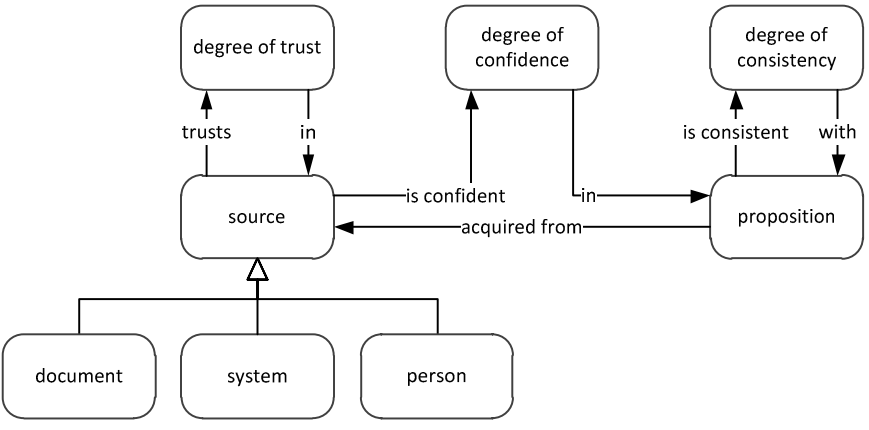


Fig. 4. Metamodel of the source and assessment of each proposition

The degrees of trust, confidence, and consistency are real numbers in the interval [-1,1]. A *degree of trust* of -1 indicates that a source does not trust another source entirely. Basically, it represents that source A believes that every proposition acquired from source B is false. Conversely, a degree of trust of 1 indicates complete trust in a certain source. The degree of confidence indicates how confident a source is of a certain proposition: -1 indicates that the source is entirely convinced that the proposition is false, whereas 1 indicates that the source believes without a doubt that the proposition is true. The same reasoning applies to the *degree of consistency* between propositions: -1 indicates that proposition A is the negation of proposition B, and 1 indicates that proposition A and B are equivalent.

These assessments allow us to reason about the propositions acquired by the sources and thus provide a more accurate depiction of the organization. It is important to highlight a special kind of proposition, one that indicates that another proposition presents a breakdown in the contextual design sense, that is, a problem found in the organization that needs to be addressed.

We propose the use of a semi-structured or frame-like representation, which allows us not only to define propositions in various levels of abstraction, but also to associate them with the information sources and to easily convert them to a semantic web representation such as RDF, to use available reasoners for making inferences.

From each proposition, other propositions may be derived following a systematic question.

5 The Knowledge Elicitation Process

Advances in Social Sciences, particularly in the last few decades, provided software engineers with a plethora of field-tested approaches to aid in the elicitation of processes, knowledge and system requirements [9, 24, 6, 27] . The generality of some of the methods, however, prevents them from being taken to their word in the IT domain. Not only it is complex, with many subtleties and peculiarities, but also presents several political and legal issues that have to be taken into account. In what follows we discuss a few knowledge elicitation techniques, pointing out to some special considerations that must be taken into account for the IT case.

The first problem one faces any new situation may be one of the most challenging one: to understand what the stakeholders are saying [7]. A common solution is the use of a glossary. The main idea behind the use of a glossary is that before modeling any domain, one has to understand the vocabulary used in it. Capturing domain language and organizing it in the glossary format not only comes at low cost, but also serves to facilitate the reuse of the knowledge about the domain [6].

The Language Extended Lexicon (LEL) [18] is an example of one such glossary. The objective of the LEL is to register the vocabulary of a given Universe of Discourse (UofD). It is based upon the following simple idea: understand the problem's language without worrying about deeply understanding the problem. The main objective of the LEL is to register signs (words or phrases) peculiar to a specific field of application. The LEL is based on a code system composed of symbols where each symbol is an entry expressed in terms of notions and behavioral responses. The notions must try to elicit the meaning of the symbol and its fundamental relations with other entries. The behavioral response must specify the connotation of the symbol in the UofD. Each symbol may also be represented by one or more aliases and will be classified as a subject a verb or an object. The construction of the LEL must be oriented by the minimum vocabulary and the circularity principles. The circularity principle prescribes the maximization of the usage of LEL symbols when describing LEL entries, while the minimal vocabulary principle prescribes the minimization of the usage of symbols exterior to the LEL when describing LEL entries. Because of the circularity principle, the LEL has a hypertextual format, which uses links to provide connections with other terms in the glossary.

The lexicon is build while conducting open-ended interviews with the stakeholders. During these meetings one tries to get a first idea of the domain, and begin to capture some initial LEL symbols. It is also important to try to identify possible documents that can be used to elicit additional symbols. The use of other types of interview and protocol analysis [9] can be particularly useful in constructing the lexicon.

5.1 Document Reading

Using existing documents, job descriptions, task descriptions and quality assurance manuals among other type of documents can be of enormous help understanding relationships and processes in the IT topology. Among the advantages are the ease of access to, usually very large, amounts of information, non-intrusiveness, and the fact that it can be done asynchronously. It can be useful in the elicitation of glossary terms

(domain language acquisition) and in building sketch process models, to be validaded with users during interviews or meetings. Document reading, however, is a time consuming task, that requires expertise in the identification of relevant information that, most of the time is sparsely scattered.

5.2 Interviews

Interviews are used in a variety of domains, and are often quite successful; see [22] for a good survey. Essentially there are two kinds of interviews: structured and non-structured. A structured or patterned interview makes use of a standard set of questions that are asked of all candidates. Organized in a similar way as a questionnaire, this kind of interview facilitates the evaluation and comparison of results. Non-structured interviews, on the other hand, leave room for digressions and comments. They are very useful in the beginning elicitation stages, when one wants to get a general feel and overall idea of the processes.

The most challenging part of applying the interview technique is, no doubt, deciding whom to interview. Projects often fail because they overlook stakeholders, so identifying the correct people is fundamental to get the right picture of an organization [1].

5.3 Stakeholder Meetings

Criticism to interviewing and the use questionnaires, which focus on individual input rather than group consensus, led to the creation of stakeholder meeting techniques. These techniques make use of customer involvement and group dynamics to accurately depict the user's view of the business needs. A good example of such technique is the Joint Applicaton Development (JAD), created in the late 70's by Chuck Morris, an IBM manager. A cousin to the focus group technique, JAD functions as a group interview. In this technique, groups are brought together to discuss system requirements. This is often done using materials such as story boards, prototypes, or product mockups, and is commonly used to get the opinions and elicit the needs of identified stakeholders. JAD meetings have the advantage of allowing more natural interactions between people than questionnaire interviews, or even open ended interviews. Besides gathering knowledge about the domain, such meetings are also very good to solve conflicts among stake holders early in the process. They have recently become popular in Requirements Engineering, especially for Information Systems applications, because of their claim to greatly accelerate the development of requirements [8]. Drawbacks are the high cost (more than monetary, the difficulty of reconciling the agenda of all stakeholders), and the fact that participants are unable to articulate tacit knowledge. Our practical experience using JAI also demonstrated that, because participants may have different status within the organization, there is a danger that some will not feel free to say what they really think, especially if it is unpopular [6].

5.4 Ethnography

Ethnography is a data capture method that immerses a researcher within an organization in an attempt to discover the actual work practices being carried out,

providing a detailed, in-depth description of everyday tasks and practice. Its aim is to focus on the users, and to discover the real world practices that are important to how they work [26].

A classical ethnographical study requires a great amount of time: data must be gathered, modeled and processed, in a process that requires full time dedication. In today's business environment such dedication is not always possible. In this light the quick and dirty ethnography was developed [25, 13]. Its aim to allow ethnography of some form to be carried out even in the most constrained projects. Similarly to the quick and dirty approach, lightweight ethnography techniques have been gaining momentum in the requirements engineering, see [27] for a complete Survey.

Goguen et al. suggest that it is often a good idea to start with an ethnographic study to uncover basic aspects of social order, such as the basic category systems used by members, the division into social groups, the goals of various social groups, typical patterns of work, how current technology is used" [9].

6 Related Work

In [14], Jahn and Lawson present alternative models of IT organizations centered on value-based IT services. They also propose recipes for transforming an IT organization from its AS-IS state to a desired TO-BE state. This work is definitely relevant to the modeling aspect of our approach, though our aim is to provide a more comprehensive and thorough model of the IT organization at least one level of detail deeper than Jahn and Lawson's.

In [5], Cross and Parker use social network analysis to draw pictures (sociograms) of organizations that reflect what the organization looks like in reality rather than representing an ideal picture of the organization. With respect to our proposed approach, Cross and Parker's work concentrates on highlighting the importance of the picture, but only gives guidelines on how to draw it through studying the organization. Our proposed approach on the other hand aims at providing a workable, practical methodology to derive such sociograms from both ethnographic studies and mining available enterprise data and documents.

The seminal works of Krackhardt et al. [16, 17] offer a variety of models that our approach proposes to build upon. In [16], Krackhardt and Carley propose a model of organizations structured around the domains of Individual, Tasks and Resources. Based on that, they define five primitive relations among these elements: Precedence, Commitment of Resources, Assignment of personnel to tasks, Network and Skills (PCANS). They envisage using the PCANS model to uncover patterns of collaboration and interdependencies within the enterprise. Our model presents aspects that are similar and compatible to PCANS, but it goes beyond it in that it introduces roles individuals play, relationships between roles rather than individuals, and makes explicit the difference between the concepts of accountability and responsibility of roles. In [17], Krackhardt and Hanson introduce the advice network, the trust network and the communication network, aspects of which are reflected in our model also. The main difference between our work and [17] is once again that not only do we propose to use questionnaires and ethnographic studies to derive the networks, but we will mine data and documents in the enterprise.

7 Concluding Remarks

In this paper we have begun to lay out the foundation for comprehensive studies of IT organizations that promise to be useful in guaranteeing a successful transformation step in IT outsourcing deals. The model and methodology presented here form the basis for our approach. We are at the beginning of our journey to support model and methodology with a suite of tools to realize the approach and make it useful in practice, ensuring that the outsourcing organization embracing our approach can drive down cost, increase margin and improve customer satisfaction.

Acknowledgments

The authors thank Hewlett-Packard (Convênio de Cooperação Técnica n.º HP-032/10) for the support to this work. Simone Barbosa and Karin Breitman also thank CNPq research grant #313031/2009-6, and the Brazilian Institute for Web Science Research (CNPq 557.128/2009-9 and FAPERJ E-26/170028/2008).

References

1. Alexander, I., Robertson, S.: Understanding project sociology by modeling stakeholders. IEEE Software 21(1), 23–27 (2004)
2. Beyer, H., Holtzblatt, K.: Contextual Design: Defining Customer-Centered Systems. Morgan Kaufmann Publishers, Inc., San Francisco (1998)
3. Brennan, K.: A Guide to the Business Analysis Body of Knowledge (Babok Guide), p. 29. International Institute of Business Analysis (2009) ISBN 0981129218
4. Carroll, J.M., Mack, R.L., Robertson, S.P., Rosson, M.B.: Binding Objects to Scenarios of Use. International Journal of Human-Computer Studies 41, 243–276 (1994)
5. Cross, R.L., Parker, A.: The Hidden Power of Social Networks: Understanding How Work Really Gets Done in Organizations. Harvard Business Press, Boston (2004)
6. Cysneiros, L.M.: Requirements Engineering in the Health Care Domain. In: RE 2002, pp. 350–356 (2002)
7. D'Souza, D.F., Will, A.C.: Objects, Components and Frameworks With UML: The Catalysis Approach. Addison-Wesley, Reading (1999)
8. Andrews, D.C.: JAD: A crucial demension for rapid applications development. Journal of Systems Management, 23–31 (March 1991)
9. Goguen, J., Linde, C.: Techniques for Requirements Elicitation. In: First International Symposium on Requirements Engineering, p. 152. IEEE Computer Society Press, Los Alamitos (1993)
10. Hertzum, M.: The importance of trust in software engineers' assessment and choice of information sources. Information and Organization 12, 1–18 (2002)
11. Hertzum, M., Andersen, H.H.K., Andersen, V., Hansen, C.B.: Trust in information sources: seeking information from people, documents, and virtual agents. Interacting with Computers 14, 575–599 (2002)
12. Holtzblatt, K., Wendell, J.B., Wood, S.: Rapid Contextual Design: A How-to guide to key techniques for user-centered design. Morgan Kaufmann Publishers, Inc., San Francisco (2005)

13. Hughes, J., O'Brien, J., Rodden, T., Rouncefield, M., Sommerville, I.: Presenting ethnography in the requirements process. In: Second IEEE International Symposium on Requirements Engineering, RE 1995, p. 27 (1995)
14. Jahn, K., Lawson, R.: Transforming IT: A Five-Step Plan for Adopting a Service Centric IT Model, Barrios (2009)
15. Kolodner, J.L., Leake, D.B.: A Tutorial Introduction to Case-Based Reasoning. In: Leake, D.B. (ed.) Case-Based Reasoning: Experiences, Lessons, and Future directions. The MIT Press, AAAI Press, Cambridge, MA (1996)
16. Krackhardt, D., Carley, K.M.: A PCANS Model of Structure in Organizations. In: 1998 International Symposium on Command and Control Research and Technology, Monterey, CA (1998)
17. Krackhardt, D., Hanson, J.R.: Informal Networks: The Company Behind the Chart. Harvard Business Review 71(4), 104–111 (1993)
18. Leake, D.B.: CBR in Context: The Present and Future. In: Leake, D.B. (ed.) Case-Based Reasoning: Experiences, Lessons, and Future directions. The MIT Press, AAAI Press, Cambridge, MA (1996)
19. Leite, J.C.S.P., Franco, A.P.M.: A Strategy for Conceptual Model Acquisition. In: Proceedings of the First IEEE International Symposium on Requirements Engineering, pp. 243–246. IEEE Computer Society Press, SanDiego (1993)
20. Lim, S.L., Quercia, D., Finkelstein, A.: StakeSource: harnessing the power of crowdsourcing and social networks in stakeholder analysis. In: Proceedings of the 32nd ACM/IEEE international Conference on Software Engineering, ICSE 2010, Cape Town, South Africa, May 01 - 08, vol. 2, pp. 239–242. ACM, New York (2010)
21. Miles, S., Groth, P., Deelman, E., Vahi, K., Mehta, G.: Provenance: The Bridge Between Experiments and Data. In: Computing in Science & Engineering, pp. 38–46 (May/June 2008)
22. Moser, C., Kalton, G.: Survey Methods in Social Investigation. Gower, England (1971)
23. Non-Functional Requirements into data model. In: 4th International Symp. on Requirements Engineering (June 1999)
24. Randal, D., Hughes, J., Shapiro, D.: Steps toward a partnership: Ethnography and system design, Draft, Lancaster University (1992)
25. Socio-Technical Systems at University of Saint Andrew: Ethnography, `http://miud.in/92E`
26. Sommerville, I.: Notes, `http://miud.in/92D`
27. Sommerville, I.: Requirements Engineering. John Wiley and Sons, Chichester (1998)
28. Tseng, S., Fogg, B.J.: Credibility and computing technology. Communications of the ACM 42(5), 39–44 (1999)

On the 2-Categorical View of Proofs

Cecilia Englander and Edward Hermann Haeusler

Departamento de Informatica PUC-Rio

Abstract. The relationship between logic and Category Theory was initiated by Lambek when viewing deductive systems as free categories with deductions as morphisms, formulas as objects and the cut-rule as composition. MacLane coherence theorems on monoidal categories showed how equality between morphism in a category resembles equality between proofs in a system with a kind of cut-elimination theorem. This raised what is nowadays known as categorical logic. Intuitionistic Natural Deduction systems are mapped into suitable categories according to *formula-as-objects* and *proofs-as-morphisms* notions of interpretation, extended to include functors as the categorical counterpart of the logical connectives. On the Proof-Theoretical side, Prawitz reductionistic conjecture plays a main role on dening an identity criteria between logical derivations. Reductions between proofs are worth knowing and representing whenever a deeper understanding of equality is present. From the 1-categorical point of view, morphisms are compared only by means of equations. This brings asymmetries into the proof-theoretical and categorical relationship. In the 70s Seely considered a 2-categorical interpretation as a solution to this problem. This article details Seely's proposal and shows how even under this broader interpretation Prawitz based identity criteria cannot be completely supported. The article also considers the recent use of structural reductions, a kind of global reduction between proofs, as a help for supporting Prawitz based identity criteria.

1 Introduction

A reduction in Proof Theory defines a relation between two derivations and therefore we would have to take into account arrows between morphisms in order to preserve symmetry. In 2-Category Theory, the set of morphisms with same source and target is seen also as a category, that is, the morphisms are seen as objects and, at another level, the morphisms between them as morphisms. To prevent confusion, the objects are called 0-cells, the morphisms 1-cells and the morphisms between morphisms 2-cells. We usually represent 1-cells by single arrows (e.g. $f\colon A \to B$) and 2-cells by double arrows (e.g. $\alpha\colon f \Rightarrow g$).

In a 2-category, given 1-cells f and g, it can happen that there exist 2-cells α and β such that $\alpha\colon f \Rightarrow g$ and $\beta\colon g \Rightarrow f$. Therefore, besides reductions, we have to take expansions into account. Let *rex* be either a reduction or an expansion step or a sequence of them. According to Prawitz's conjecture [5], we say that two derivations Π and Ψ represent the same derivation if either $\Pi \rhd \Psi$ or $\Psi \rhd \Pi$,

T. Margaria and B. Steffen (Eds.): ISoLA 2010, Part I, LNCS 6415, pp. 502–518, 2010.

where $\rhd$ is a rex. Derivations may be reduced and expanded as shown in [5]. We also consider reductions and expansions for derivations with at least one occurrence of $\bot$. Thus, in order to have a better picture of Prawitz identity criteria in categorical terms we have to represent reductions, derivations and formulas (types) inside a sole category. Seely [7] uses 2-Category to figure out the three components involved in analyzing proofs/derivations. Here we follow him and go a bit further when we consider the implications of having a interpretation for the $\bot$ and the inclusion of global reductions (structural reductions) in this scenario.

Throughout this article we work with a deduction system where the following three properties hold: let $\rhd$ be a rex. Then

0) If $\Pi \rhd \Pi'$ and $\Pi' \rhd \Pi''$, then $\Pi \rhd \Pi''$;

1) If $\Pi(X) \rhd \Pi'(X)$, then for all Σ, $\Pi(\Sigma) \rhd \Pi'(\Sigma)$;

2) If $\Sigma \rhd \Sigma'$ then $\Pi(\Sigma) \rhd \Pi(\Sigma')$

where $\Pi(X)$ means that X is a hypothesis of Π and $\Pi(\Sigma)$ means that every hypothesis of Π that has the shape of the conclusion of Σ is replaced by Σ. Property (0) means that rex are transitive and properties (1) and (2) means that we can either reduce a derivation and then apply substitution or apply substitution and then reduce the resulting derivation.

It is natural to try to represent reductions explicitly as 2-cells instead of having the isomorphism between arrows (1-cells). More precisely, consider derivations $\Pi_i \colon \beta[x : \alpha]$[1] and $\Psi_i \colon \gamma[x : \beta]$, $i = 1, 2, 3$. Consider also the reductions ρ_1 and ρ_2 from Π_1 to Π_2 and from Π_2 to Π_3 respectively and the reductions τ_1 and τ_2 from Ψ_1 to Ψ_2 and from Ψ_2 to Ψ_3 respectively. We have then the following diagram:

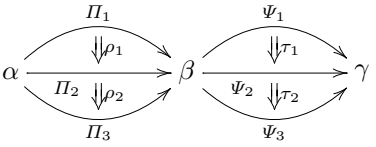
$$\begin{array}{ccccc} \alpha & & \alpha & & \alpha \\ \Pi_1 & \rhd_{\rho_1} & \Pi_2 & \rhd_{\rho_2} & \Pi_3 \\ \beta & & \beta & & \beta \\ \Psi_1 & \rhd_{\tau_1} & \Psi_2 & \rhd_{\tau_2} & \Psi_3 \\ \gamma & & \gamma & & \gamma \end{array}$$

where a derivation $\Psi \circ \Pi \colon \gamma[x \colon \alpha]$ is achieved by substituting every occurrence of β as a premise in Ψ_i for $\Pi_i \colon \beta[x : \alpha]$.

We have several possibilities to reduce the leftmost derivation: we can, for instance, apply ρ_1 followed by ρ_2 followed by τ_1 followed by τ_2 or ρ_1 followed by τ_1, followed by ρ_2 followed by τ_2. Any choice should produce the same result, namely, the rightmost derivation.

[1] For typographical reasons sometimes we use the λ-Calculus notation $\Pi \colon \beta[x \colon \alpha]$ instead of $\begin{array}{c} \alpha \\ \Pi \\ \beta \end{array}$.

If we represent the formulas of the above diagram by 0-cells, the derivations by 1-cells and the rex by 2-cells, we come to the 2-categorical diagram



By the interchange law of 2-Category Theory, $(\tau_2 \cdot \tau_1);(\rho_2 \cdot \rho_1) = (\tau_2;\rho_2) \cdot (\tau_1;\rho_1)$ where "$\cdot$" represents vertical composition and ";" represents horizontal composition. Note that this diagram corresponds exactly to the three properties presented before, meaning that there exists a direct correspondence between 2-categorical and proof-theoretical composition. According to this approach, we would say that two 1-cells Π and Ψ are equivalent if there exist a 2-cell from Π to Ψ.

In this article we assume that the reader is aware of 2-Category terminology and main definitions. In what follows we only present the 2-categorical definitions that are used in this article.

2 Some 2-Categorical Notions

2-Functor: Given two categories $\mathcal{A}$ and $\mathcal{B}$, a functor F from $\mathcal{A}$ to $\mathcal{B}$ is a function that takes to $\mathcal{B}$ the categorical structure of $\mathcal{A}$. The 2-functor definition subsumes the functor definition (with equalities been replaced by natural isomorphisms) with the addition of preserving the structure of the 2-cells, i. e.:

(i) if A is an $\mathcal{A}$-object, then $F(A)$ is a $\mathcal{B}$-object;
(ii) if $f\colon A \to B$, then $F(f)\colon F(A) \to F(B)$;
(iii) for every $\mathcal{A}$-object A, $F(id_A) \cong id_{F(A)}$;
(iv) if $h = f \circ g$, then $F(h) \cong F(f) \circ F(g)$;
(v) if $\alpha\colon f \Rightarrow g$, then $F(\alpha)\colon F(f) \Rightarrow F(g)$;
(vi) for every 1-cell f, $F(id_f) = id_{F(f)}$;
(vii) if $\gamma = \alpha \cdot \beta$, then $F(\gamma) = F(\alpha) \cdot F(\beta)$;
(viii) if $\gamma = \alpha;\beta$, then $F(\gamma) = F(\alpha);F(\beta)$.

Let $Hom(X,Y)$ be the set of arrows that go from the 0-cell X to the 0-cell Y and $Hom_{Cat}(X,Y)$ the category formed by $Hom(X,Y)$. A 1-categorical product is determined by a natural isomorphism between $Hom(X,\ A \times B)$ and $Hom(X,A) \otimes Hom(X,B)$, so it is natural to define 2-categorical product as coming from the natural isomorphism between the categories $Hom_{Cat}(X, A \times B)$ and $Hom_{Cat}(X,A) \otimes Hom_{Cat}(X,B)$. Hence, a *2-categorical product*, or simply 2-product, of two 0-cells A and B is a 0-cell $A \times B$ together with two projection arrows $\pi_1\colon A \times B \to A$ and $\pi_2\colon A \times B \to B$ such that for any $f\colon X \to A$ and $g\colon X \to B$, there exist $h\colon X \to A \times B$ and isomorphisms $\pi_1 \circ h \cong f$ and $\pi_2 \circ h \cong g$ and such that for all $k\colon X \to A \times B$ and 2-cells $\alpha\colon \pi_1 \circ h \Rightarrow \pi_1 \circ k$ and $\beta\colon \pi_2 \circ h \Rightarrow \pi_2 \circ k$ there exist a unique $\gamma\colon h \Rightarrow k$ such that $id_{\pi_1};\gamma = \alpha$ and $id_{\pi_2};\gamma = \beta$.

Given 2-categories $\mathcal{A}$ and $\mathcal{B}$, if F is a 2-functor from $\mathcal{A}$ to $\mathcal{B}$ and G is a 2-functor from $\mathcal{B}$ to $\mathcal{A}$, then a 0-cell A in $\mathcal{A}$ and a 0-cell B in $\mathcal{B}$ define the categories $Hom_{Cat}(FA, B)$ and $Hom_{Cat}(A, GB)$.

The word *lax* preceding a notion means that the given notion is "weakened": the isomorphisms in the definition are replaced by simple arrows, e.g., if F is a 2-functor and f and g are arrows, a lax 2-functor would, instead of $F(f) \circ F(g) \cong F(f \circ g)$, have either $F(f \circ g) \to F(f) \circ F(g)$ or $F(f) \circ F(g) \to F(f \circ g)$, weakening the definition of 2-functor.

Given two 2-categories $\mathcal{A}$ and $\mathcal{B}$ and lax 2-functors $F\colon \mathcal{A} \to \mathcal{B}$ and $G\colon \mathcal{B} \to \mathcal{A}$, we say that F is lax right 2-adjoint to G when, for every 0-cell A in $\mathcal{A}$ and B in $\mathcal{B}$, there exist arrows K_{AB} from $Hom_{Cat}(F(A), B)$ to $Hom_{Cat}(A, G(B))$ and L_{AB} from $Hom_{Cat}(A, G(B))$ to $Hom_{Cat}(F(A), B)$ such that those arrows define the following natural transformations:

1. $L_{A'B} \circ Hom(f, G(B)) \Rightarrow Hom(F(f), B) \circ L_{AB}$;
2. $L_{A'B'} \circ Hom(A', G(g)) \Rightarrow Hom(F(A'), g) \circ L_{A'B}$;
3. $Hom(f, G(B)) \circ K_{AB} \Rightarrow K_{A'B} \circ Hom(F(f), B)$;
4. $Hom(A', G(g)) \circ K_{A'B} \Rightarrow K_{A'B'} \circ Hom(F(A'), g)$;
5. $\alpha\colon L_{AB} \circ K_{AB} \Rightarrow id_{Hom_{Cat}(F(A),B)}$;
6. $\beta\colon id_{Hom_{Cat}(A,G(B))} \Rightarrow K_{AB} \circ L_{AB}$.

where $f\colon A' \to A$ is a 1-cell in $\mathcal{A}$ and $g\colon B \to B'$ is a 1-cell in $\mathcal{B}$. We also have that $(id_{K_{AB}}; \alpha) \cdot (\beta; id_{K_{AB}}) = id_{K_{AB}}$ and $(\alpha; id_{L_{AB}}) \cdot (id_{L_{AB}}; \beta) = id_{L_{AB}}$ hold.

If we had the direction of the arrows 1 to 6 going the other way around and that $(\beta; id_{K_{AB}}) \cdot (id_{K_{AB}}; \alpha) = id_{K_{AB}}$ and $(id_{L_{AB}}; \beta) \cdot (\alpha; id_{L_{AB}}) = id_{L_{AB}}$ hold, then, instead of lax right 2-adjointness, we would have lax left 2-adjointness. This definition can be better understood with the help of the following diagram:

$$\begin{array}{ccc}
Hom_{Cat}(FA, B) & \underset{L_{AB}}{\overset{K_{AB}}{\rightleftarrows}} & Hom_{Cat}(A, GB) \\
\downarrow{\scriptstyle Hom(Ff, B)} & & \downarrow{\scriptstyle Hom(f, GB)} \\
Hom_{Cat}(FA', B) & \underset{L_{A'B}}{\overset{K_{A'B}}{\rightleftarrows}} & Hom_{Cat}(A', GB) \\
\downarrow{\scriptstyle Hom(FA', g)} & & \downarrow{\scriptstyle Hom(A', Gg)} \\
Hom_{Cat}(FA', B') & \underset{L_{A'B'}}{\overset{K_{A'B'}}{\rightleftarrows}} & Hom_{Cat}(A', GB')
\end{array}$$

Fig. 1. natural transformation's diagram

3 2-Category and Proof Theory

Seely [7] states that conjunction is lax right 2-adjunction to the diagonal functor, that conjunction is lax left 2-adjunction to the diagonal functor and that implication is neither of them, but has some of the properties of the two forms of 2-adjunction. Seely uses the expression lax 2-adjunction instead of lax right 2-adjointness and rax 2-adjunction instead of lax left 2-adjointness.

In order to deal with our 2-categorical approach to proofs we need to introduce a combination of reductions between proofs: given a derivation of the form $\dfrac{\begin{matrix}\Pi_1 & \Pi_2\\ A & B\end{matrix}}{C}$, if $\Pi_1 \rhd_\alpha \Pi_1'$ and $\Pi_2 \rhd_\beta \Pi_2'$, then $\alpha \mid \beta$ is the rex that takes $\dfrac{\begin{matrix}\Pi_1 & \Pi_2\\ A & B\end{matrix}}{C}$ to $\dfrac{\begin{matrix}\Pi_1' & \Pi_2'\\ A & B\end{matrix}}{C}$ by first reducing Π_1 and then Π_2. Note that reducing Π_2 and then Π_1 should produce the same result.

Proposition 31. *For every pair of reductions* (α_1, α_2) *and* (β_1, β_2), $(\alpha_1 \mid \alpha_2) \cdot (\beta_1 \mid \beta_2) = (\alpha_1 \cdot \beta_1) \mid (\alpha_2 \cdot \beta_2)$.

3.1 Conjunction

As categorical product is related to conjunction [3], it is natural to try to achieve a similar result in 2-Category Theory: given two 0-cells A and B, if we represent the formula $A \wedge B$ by the product $A \times B$ and the deduction rules $\dfrac{A \wedge B}{A}$ and $\dfrac{A \wedge B}{B}$ by the projection arrows $\pi_1 \colon A \times B \to A$ and $\pi_2 \colon A \times B \to B$ respectively, h would be represented by $\dfrac{\begin{matrix}C & C\\ f & g\\ A & B\end{matrix}}{A \wedge B}$. In order to $A \wedge B$ be a 2-categorical product, we would have $\pi_1 \circ h$ and f isomorphic and hence

$$\dfrac{\dfrac{\begin{matrix}C & C\\ f & g\\ A & B\end{matrix}}{A \wedge B}\, h}{A}\, \pi_1 \quad \rhd_\alpha \quad \begin{matrix}C\\ f\\ A\end{matrix} \quad . \tag{1}$$

However, β does not represent any meaningful proof-theoretical relation. Such a reason is enough to make us realize that conjunction does not have the same structure of 2-categorical product.

As we have only one of the relations in (1), the one that represents $\wedge$-reduction, we can try to relate conjunction to lax 2-product. Let k be an arrow from C to $A \times B$. Then the 2-cells α and β from $\pi_1 \circ h$ to $\pi_1 \circ k$ and from $\pi_2 \circ h$ to $\pi_2 \circ k$ respectively can be represented by the two rex

$$\dfrac{\begin{matrix}C\\ h\\ A \wedge B\end{matrix}}{A}\, \pi_1 \quad \rhd_\alpha \quad \dfrac{\begin{matrix}C\\ k\\ A \wedge B\end{matrix}}{A}\, \pi_1 \qquad\qquad \dfrac{\begin{matrix}C\\ h\\ A \wedge B\end{matrix}}{B}\, \pi_2 \quad \rhd_\beta \quad \dfrac{\begin{matrix}C\\ k\\ A \wedge B\end{matrix}}{B}\, \pi_2$$

To achieve the wanted relation, we would have to show that there exists only one $\gamma \colon h \Rightarrow k$ such that $id_{\pi_1}; \gamma = \alpha$ and $id_{\pi_2}; \gamma = \beta$. As 2-cells are defined up to isomorphism, this unicity cannot be proved. Another way of 2-categorically represent rex is as a preorder relation, that is, by representing every rex by the same 2-cell. Thus the unicity of γ is trivially satisfied and, in this picture, conjunction can be seen as lax 2-product.

Seely states that conjunction is lax right 2-adjunction to the diagonal functor. We show the proof of this fact in detail. We separate the definition of lax right 2-adjunction in four parts on an attempt to make the proof easier to read:

first part *given the 2-categories $\mathcal{PT}$ and $\mathcal{PT} \times \mathcal{PT}$, where $\mathcal{PT}$ is the 2-category whose 0-cells are formulas, 1-cells are derivations and 2-cells are reductions, we show that both $\Delta\colon \mathcal{PT} \to \mathcal{PT} \times \mathcal{PT}$ and $\wedge\colon \mathcal{PT} \times \mathcal{PT} \to \mathcal{PT}$ are lax 2-functors.*

Lemma 3.11. *$\Delta\colon \mathcal{PT} \to \mathcal{PT} \times \mathcal{PT}$ such that $\Delta(A) = (A, A)$, $\Delta(f) = (f, f)$ and $\Delta(\alpha) = (\alpha, \alpha)$ for every formula A, derivation f and reduction α, is a lax 2-functor.*

Lemma 3.12. *$\wedge\colon \mathcal{PT} \times \mathcal{PT} \to \mathcal{PT}$ such that $\wedge(A) = A_1 \wedge A_2$, $\wedge(f) = f_1 \wedge f_2 =$*

$$\dfrac{\dfrac{A_1 \wedge A_2}{A_1}\ f_1\ \dfrac{A'_1}{}}{A'_1 \wedge A'_2} \quad \dfrac{\dfrac{A_1 \wedge A_2}{A_2}\ f_2\ \dfrac{A'_2}{}}{}$$

$= \langle f_1 \circ \pi_1, f_2 \circ \pi_2 \rangle$, where $\langle f, g \rangle$ is the canonical morphism from A to $B \times C$, with f going from A to B and g going from A to C and $\wedge(\alpha) = (\alpha_1; id_{\pi_1}) \mid (\alpha_2; id_{\pi_2})$ for every formula $A = (A_1, A_2)$, derivation $f = (f_1, f_2)$, $f_i\colon A_i \to A'_i$ and rex $\alpha = (\alpha_1, \alpha_2)$, is a lax 2-functor.

Proof

(iii) *Given a formula A, both $id_{A_1} \wedge id_{A_2}$ and $id_{A_1 \wedge A_2}$ are the derivation $A_1 \wedge A_2$;*

(iv) *Given derivations $f\colon A \to B$ and $g\colon B \to C$, $\wedge f \circ \wedge g$reduces to $\wedge(f \circ g)$.*

(vi) *Given a derivation $f\colon A \to B$, $\wedge(id_f) = (id_{f_1}; id_{\pi_1}) \mid (id_{f_2}; id_{\pi_2})$ is the identity reduction that goes from $\wedge f$ to $\wedge f$.*

(vii) *Given reductions $\alpha\colon f \Rightarrow g$ and $\beta\colon g \Rightarrow h$,*

$$\wedge\alpha \cdot \wedge\beta =$$
$$((\alpha_1; id_{\pi_1}) \mid (\alpha_2; id_{\pi_2})) \cdot ((\beta_1; id_{\pi_1}) \mid (\beta_2; id_{\pi_2})) = \textit{(Proposition 31)}$$
$$((\alpha_1; id_{\pi_1}) \cdot (\beta_1; id_{\pi_1})) \mid ((\alpha_2; id_{\pi_2}) \cdot (\beta_2; id_{\pi_2})) = \textit{(2-comp. property)}$$
$$((\alpha_1 \cdot \beta_1); (id_{\pi_1} \cdot id_{\pi_1})) \mid ((\alpha_2 \cdot \beta_2); (id_{\pi_2} \cdot id_{\pi_2})) = \textit{(identities comp.)}$$
$$((\alpha_1 \cdot \beta_1); id_{\pi_1}) \mid ((\alpha_2 \cdot \beta_2); id_{\pi_2}) = \wedge(\alpha \cdot \beta)$$

(viii) *We only remark that applying $\wedge\alpha$ and $\wedge\beta$ followed by $\wedge$-reduction is the same as applying $\wedge$-reduction followed by $\wedge\alpha$ and $\wedge\beta$.*

second part *we show that there exists an arrow K_{AB} from $Hom_{Cat}(\Delta A, B)$ to $Hom_{Cat}(A, \wedge B)$ and an arrow L_{AB} from $Hom_{Cat}(A, \wedge B)$ to $Hom_{Cat}(\Delta A, B)$, for every 0-cell A in $\mathcal{PT}$ and B in $\mathcal{PT} \times \mathcal{PT}$.*

Lemma 3.13. *There exists a functor K_{AB} from $Hom_{Cat}(\Delta A, B)$ to $Hom_{Cat}(A, \wedge B)$.*

Let us define K_{AB} as a function that takes a pair of derivations from $Hom_{Cat}(\Delta A, B)$ and applies $\wedge$-int, i.e., for every $\Pi = (\Pi_1, \Pi_2) \in Hom_{Cat}(\Delta A, B)$,

$$K_{AB}(\Pi) = \dfrac{\dfrac{A}{\Pi_1}\ B_1 \quad \dfrac{A}{\Pi_2}\ B_2}{B_1 \wedge B_2} \in Hom_{Cat}(A, \wedge B).$$

Given a rex $\alpha = (\alpha_1, \alpha_2)\colon \Pi \rhd \Pi'$ in $Hom_{Cat}(\Delta A, B)$, $K_{AB}(\alpha) = \alpha_1 \mid \alpha_2$.

Let us prove that K_{AB} thus defined is a functor (p. 504):

Proof
(i) Comes from the definition of K_{AB};
(ii) If $\Pi_1 \rhd_{\alpha_1} \Pi_1'$ and $\Pi_2 \rhd_{\alpha_2} \Pi_2'$, then $K_{AB}(\Pi) \rhd_{\alpha_1|\alpha_2} K_{AB}(\Pi')$
(iii) Given a derivation Π,$K_{AB}(\Pi) \rhd_{id_{\Pi_1}|id_{\Pi_1}} K_{AB}(\Pi)$ and thus $id_{K_{AB}(\Pi)} = K_{AB}(id_\Pi)$: $K_{AB}(\Pi) \rhd K_{AB}(\Pi)$
(iv) Comes from proposition 31.

Lemma 3.14. *There exists a functor L_{AB} from $Hom_{Cat}(A, \wedge B)$ to $Hom_{Cat}(\Delta A, B)$.*

For every derivation Ψ in $Hom_{Cat}(A, \wedge B)$, put $L_{AB}(\Psi)$ as being the pair $\left(\begin{array}{c} A \\ \Psi \\ \hline B_1 \wedge B_2 \\ \hline B_1 \end{array} , \begin{array}{c} A \\ \Psi \\ \hline B_1 \wedge B_2 \\ \hline B_2 \end{array} \right)$ and, for every rex α: $\Psi \rhd \Psi'$ in $Hom_{Cat}(A, \wedge B)$, $L_{AB}(\alpha) = (id_{\pi_1}; \alpha, id_{\pi_2}; \alpha)$.

Let us prove that L_{AB} thus defined is a functor:

Proof
(i) and (ii) come from the definition of L_{AB};
(iii) Given a derivation Ψ, $L_{AB}(id_\Psi) = (id_{\pi_1}; id_\Psi, id_{\pi_2}; id_\Psi)$, which is the identity reduction that goes from $L_{AB}(\Psi)$ to $L_{AB}(\Psi)$.
(iv) Given rex α and β,

$$\begin{aligned} L_{AB}(\alpha) \cdot L_{AB}(\beta) &= \\ (id_{\pi_1}; \alpha, id_{\pi_2}; \alpha) \cdot (id_{\pi_1}; \beta, id_{\pi_2}; \beta) &= \\ ((id_{\pi_1}; \alpha) \cdot (id_{\pi_1}; \beta), (id_{\pi_2}; \alpha) \cdot (id_{\pi_2}; \beta)) &= \text{(2-comp. property)} \\ ((id_{\pi_1} \cdot id_{\pi_1}); (\alpha \cdot \beta), (id_{\pi_2} \cdot id_{\pi_2}); (\alpha \cdot \beta)) &= \text{(comp. of identities)} \\ (id_{\pi_1}; (\alpha \cdot \beta), id_{\pi_2}; (\alpha \cdot \beta)) &= L_{AB}(\alpha \cdot \beta) \end{aligned}$$

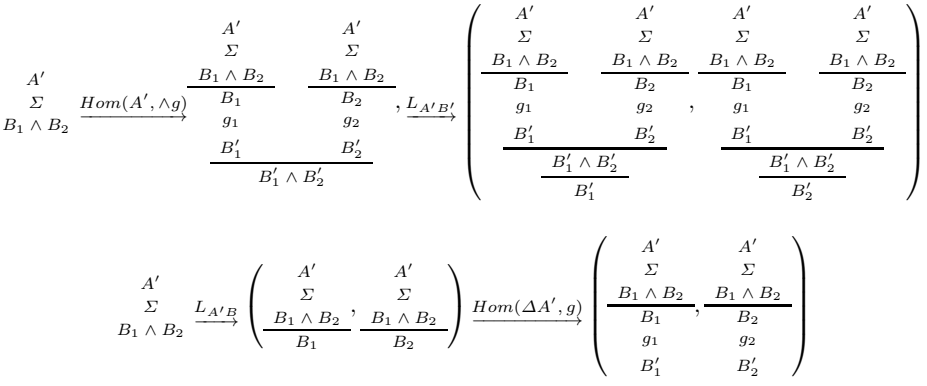
third part *we show that the arrows 1 to 6 in the definition define the natural transformations correspondent to this definition.*

Note that we can define a category whose objects are functors and whose arrows are natural transformations. Therefore, to show that the natural transformations hold, we only need to show that they correspond to a rex.

Given derivations $g = (g_1, g_2)$ in $\mathcal{A}$, g_1: $B_1'[x\colon B_1]$ and g_2: $B_2'[x\colon B_2]$, and f: $A[x\colon A']$ in $\mathcal{B}$, we have the following natural transformations (put $F = \Delta$ and $G = \wedge$ in figure 1):

1. $L_{A'B} \circ Hom(f, \wedge B) \Rightarrow Hom(\Delta f, B) \circ L_{AB}$: given a derivation Σ: $B_1 \wedge B_2[x\colon A]$, $L_{A'B} \circ Hom(f, \wedge B)(\Sigma) = Hom(\Delta f, B) \circ L_{AB}(\Sigma)$ and this natural transformation corresponds to the identity reduction.

2. $L_{A'B'} \circ Hom(A', \wedge g) \Rightarrow Hom(\Delta A', g) \circ L_{A'B}$: given a derivation Σ: $B_1 \wedge B_2[x\colon A']$,



We have that $L_{A'B'} \circ Hom(A', \wedge g)(\Sigma)$ reduces to $Hom(\Delta A', g) \circ L_{A'B}(\Sigma)$ and this natural transformation corresponds to a pair of $\wedge$-reductions.

3. $Hom(f, \wedge B) \circ K_{AB} \Rightarrow K_{A'B} \circ Hom(\Delta f, B)$: given a pair of derivations $\Sigma = (\Sigma_1 \colon B_1[x \colon A], \Sigma_2 \colon B_2[x \colon A])$, $Hom(f, \wedge B) \circ K_{AB}(\Sigma) = K_{A'B} \circ Hom(\Delta f, B)(\Sigma)$ and this natural transformation corresponds to the identity reduction.

4. $Hom(A', \wedge g) \circ K_{A'B} \Rightarrow K_{A'B'} \circ Hom(\Delta A', g)$: given a pair of derivations $\Sigma = (\Sigma_1 \colon B_1[x \colon A'], \Sigma_2 \colon B_2[x \colon A'])$, we have that $Hom(A', \wedge g) K_{A'B}(\Sigma)$ $\wedge$-reduces to $K_{A'B'}\ Hom(\Delta A', g)\ (\Sigma)$ and this natural transformation corresponds to $\wedge$-red$|$ $\wedge$-red.

5. $\alpha \colon L_{AB} \circ K_{AB} \Rightarrow id_{Hom_{Cat}(\Delta A, B)}$: given a pair of derivations $\Sigma = (\Sigma_1 \colon B_1[x \colon A], \Sigma_2 \colon B_2[x \colon A])$, $L_{AB}K_{AB}(\Sigma)$ $\wedge$-reduces to $id_{Hom_{Cat}(A, \wedge B)}(\Sigma)$ and this natural transformation corresponds to a pair of $\wedge$-reductions.

6. $\beta \colon id_{Hom_{Cat}(A, \wedge B)} \Rightarrow K_{AB} \circ L_{AB}$: given a derivation $\Sigma \colon B_1 \wedge B_2[x \colon A]$, $id_{Hom_{Cat}(A, \wedge B)}\ (\Sigma)$ $\wedge$-expands to $K_{AB}L_{AB}(\Sigma)$ and this natural transformation corresponds to $\wedge$-expansion.

fourth part *finally, we show that* $(id_{K_{AB}}; \alpha) \cdot (\beta; id_{K_{AB}}) = id_{K_{AB}}$ *and* $(\alpha; id_{L_{AB}}) \cdot (id_{L_{AB}}; \beta) = id_{L_{AB}}$.

As $\alpha \colon L_{AB} \circ K_{AB} \Rightarrow id_{Hom_{Cat}(\Delta A, B)}$ and $\beta \colon id_{Hom_{Cat}(A, \wedge B)} \Rightarrow K_{AB} \circ L_{AB}$, we have that:

1. $(id_{K_{AB}}; \alpha) \cdot (\beta; id_{K_{AB}}) = id_{K_{AB}}$

$\beta; id_{K_{AB}}$ goes from $id_{Hom_{Cat}(\Delta A, B)} \circ K_{AB} = K_{AB}$ to $(K_{AB} \circ L_{AB}) \circ K_{AB}$ and $id_{K_{AB}}; \alpha$ goes from $K_{AB} \circ (L_{AB} \circ K_{AB})$ to $K_{AB} \circ id_{Hom_{Cat}(\Delta A, B)} = K_{AB}$ therefore their composition goes from K_{AB} to K_{AB}.

2. $(\alpha; id_{L_{AB}}) \cdot (id_{L_{AB}}; \beta) = id_{L_{AB}}$

$id_{L_{AB}}; \beta$ goes from $L_{AB} \circ id_{Hom_{Cat}(A, \wedge B)} = L_{AB}$ to $L_{AB} \circ (K_{AB} \circ L_{AB})$ and $\alpha; id_{L_{AB}}$ goes from $(L_{AB} \circ K_{AB}) \circ L_{AB}$ to $id_{Hom_{Cat}(\Delta A, B)} \circ L_{AB} = L_{AB}$ therefore their composition goes from L_{AB} to L_{AB}.

3.2 Disjunction

Prawitz [6] defined the following permutation reduction

$$\dfrac{\dfrac{\dfrac{\Pi_1}{A \vee B} \quad \dfrac{\Pi_2}{C} \quad \dfrac{\Pi_3}{C}}{C} \quad \Pi_4}{D} \;\triangleright\; \dfrac{\dfrac{\Pi_1}{A \vee B} \quad \dfrac{\dfrac{\Pi_2}{C} \quad \Pi_4}{D} \quad \dfrac{\dfrac{\Pi_3}{C} \quad \Pi_4}{D}}{D}$$

where the lowest occurrence of C is a major premise, there is at least one occurrence of C in the sequence that is the conclusion of an introduction rule and Π_4 may be empty.

In [3], Mann shows that $\vee$-reduction comes from the commutativity of the coproduct diagram, but for $\vee$-expansion, from the uniqueness of the factorization, he concludes that

$$\dfrac{A \vee B \quad \dfrac{[A]}{A \vee B} \quad \dfrac{[B]}{A \vee B}}{\begin{array}{c} A \vee B \\ \Pi \\ C \end{array}} = \dfrac{A \vee B \quad \dfrac{[A]}{A \vee B} \quad \dfrac{[B]}{A \vee B}}{\begin{array}{c} \Pi \\ C \end{array} \quad \begin{array}{c} \Pi \\ C \end{array}} \;.$$

However, if there is an application of $\rightarrow$-int in Π that discharges the major premise of the $\vee$-el (i.e., the formula $A \vee B$), then there is no derivation of the form of the right hand side. One way of solving this issue is by considering, instead of Prawitz's permutation, the following reduction:

$$\dfrac{\begin{array}{c} \\ \Pi_1 \\ A \vee B \end{array} \quad \begin{array}{c} [A] \\ \Pi_2 \\ C \end{array} \quad \begin{array}{c} [B] \\ \Pi_3 \\ C \end{array}}{\begin{array}{c} C \\ \Pi_4 \\ D \end{array}} \;\triangleright\; \dfrac{\begin{array}{c} \\ \Pi_1 \\ A \vee B \end{array} \quad \begin{array}{c} [A] \\ \Pi_2 \\ C \\ \Pi_4 \\ D \end{array} \quad \begin{array}{c} [B] \\ \Pi_3 \\ C \\ \Pi_4 \\ D \end{array}}{D}$$

where no rule in Π_4 discharges any hypothesis of Π_1. We are going to call this reduction *MDP*. Note that this reduction is more general than the permutation reduction defined by Prawitz. For instance, the lowest occurrence of C does not need to be a major premise.

Seely also used this other form of permutation to conclude that disjunction is lax left 2-adjoint to the diagonal functor. We also present the definition of lax left 2-adjunction in four parts on an attempt to make the proof easier to read. The proof is as follows:

first part *given the 2-categories $\mathcal{PT}$ and $\mathcal{PT} \times \mathcal{PT}$, we show that $\vee\colon \mathcal{PT} \times \mathcal{PT} \rightarrow \mathcal{PT}$ is a lax 2-functor (that $\Delta\colon \mathcal{PT} \rightarrow \mathcal{PT} \times \mathcal{PT}$ is a lax 2-functor is proved in the preceding section).*

Lemma 3.21. *$\vee\colon \mathcal{PT} \times \mathcal{PT} \rightarrow \mathcal{PT}$ such that $\vee(A) = A_1 \vee A_2$, $\vee(f) = f_1 \vee f_2 = [\iota_1 \circ f_1, \iota_2 \circ f_2]$, where $[f, g]$ is the canonical*

$$\dfrac{A'_1 \vee A'_2 \quad \dfrac{\begin{array}{c} [A'_1] \\ f_1 \\ A_1 \end{array}}{A_1 \vee A_2} \quad \dfrac{\begin{array}{c} [A'_2] \\ f_2 \\ A_2 \end{array}}{A_1 \vee A_2}}{A_1 \vee A_2}$$

morphism from A to $B + C$, with f going from B to A and g going from C to A and $\vee(\alpha) = id_{\vee A'} \mid ((id_{\iota_1}; \alpha_1) \mid (id_{\iota_2}; \alpha_2))$, for every formula $A = (A_1, A_2)$, derivation $f = (f_1, f_2), f_i \colon A'_i \to A_i$ and reduction $\alpha = (\alpha_1, \alpha_2)$, is a lax 2-functor.

Proof

(iii) *Given a formula A, both $id_{A_1} \vee id_{A_2}$ and $id_{A_1 \vee A_2}$ are the derivation $A_1 \vee A_2$.*

(iv) *Given derivations $f \colon A \to B$ and $g \colon B \to C$, $\vee f \circ \vee g$ reduces to $\vee(f \circ g)$.*

(vi) *Given a derivation f, $\vee(id_f) = id_{\vee A} \mid ((id_{\iota_1}; id_{f_1}) \mid (id_{\iota_2}; id_{f_2}))$ which is the identity reduction that goes from $\vee f$ to $\vee f$.*

(viii) *We only remark that applying $\vee\alpha$ and $\vee\beta$ followed by $\vee$-reduction is the same as applying $\vee$-reduction followed by $\vee\alpha$ and $\vee\beta$.*

second part *we show that there exists an arrow L_{AB} from $Hom_{Cat}(A, \Delta B)$ to $Hom_{Cat}(\vee A, B)$ and an arrow K_{AB} from $Hom_{Cat}(\vee A, B)$ to $Hom_{Cat}(A, \Delta B)$ for every 0-cell A in $\mathcal{PT}$ and B in $\mathcal{PT} \times \mathcal{PT}$.*

Lemma 3.22. *There exists a functor $K_{AB} \colon Hom_{Cat}(\vee A, B) \to Hom_{Cat}(A, \Delta B)$*

Let us define K_{AB} as the function that composes every derivation $\Pi \in Hom_{Cat}(\vee A, B)$ with $\vee$-int, i.e., $K_{AB}(\Pi) = \left(\dfrac{\dfrac{A_1}{A_1 \vee A_2}}{\begin{matrix}\Pi \\ B\end{matrix}}, \dfrac{\dfrac{A_2}{A_1 \vee A_2}}{\begin{matrix}\Pi \\ B\end{matrix}} \right)$ and for every $\alpha \in Hom_{Cat}(\vee A, B)$, $K_{AB}(\alpha) = (\alpha; id_{\iota_1}, \alpha; id_{\iota_2})$.

Proof. (iii) Given a derivation Π, $K_{AB}(id_\Pi) = (id_\Pi; id_{\iota_1}, id_\Pi; id_{\iota_2})$ which is the identity reduction that goes from $K_{AB}(\Pi)$ to $K_{AB}(\Pi)$.
(iv) Given reductions α and β,

$$
\begin{aligned}
K_{AB}(\alpha) \cdot K_{AB}(\beta) &= (\alpha; id_{\iota_1}, \alpha; id_{\iota_2}) \cdot (\beta; id_{\iota_1}, \beta; id_{\iota_2}) \\
((\alpha; id_{\iota_1}) \cdot (\beta; id_{\iota_1}), (\alpha; id_{\iota_2}) \cdot (\beta; id_{\iota_2})) &= \text{(2-comp. property)} \\
((\alpha \cdot \beta); (id_{\iota_1} \cdot id_{\iota_1}), (\alpha \cdot \beta); (id_{\iota_2} \cdot id_{\iota_2})) &= \text{(comp. of identities)} \\
((\alpha \cdot \beta); id_{\iota_1}, (\alpha \cdot \beta); id_{\iota_2}) &= K_{AB}(\alpha \cdot \beta)
\end{aligned}
$$

Lemma 3.23. *There exists a functor $L_{AB} \colon Hom_{Cat}(A, \Delta B) \to Hom_{Cat}(\vee A, B)$*

Let us define L_{AB} as the function that takes every $\Pi = (\Pi_1, \Pi_2) \in Hom_{Cat}(A, \Delta B)$ to $L_{AB}(\Pi) = \dfrac{\begin{matrix}[A_1] & [A_2] \\ \Pi_1 & \Pi_2 \\ A_1 \vee A_2 & B \quad B\end{matrix}}{B}$ and for every $\alpha = (\alpha_1, \alpha_2) \colon \Pi \rhd \Pi'$, $L_{AB}(\alpha) = id_{A_1 \vee A_2} \mid (\alpha_1 \mid \alpha_2)$.

Proof. (iii) Given a derivation Π, $L_{AB}(id_\Pi) = id_{A_1 \vee A_2} \mid (id_{\Pi_1} \mid id_{\Pi_2})$ which is the identity reduction that goes from $L_{AB}(\Pi)$ to $L_{AB}(\Pi)$.
(iv) Given reductions α and β,

$$
\begin{aligned}
L_{AB}(\alpha) \cdot L_{AB}(\beta) &= \\
(id_{A_1 \vee A_2} \mid (\alpha_1 \mid \alpha_2)) \cdot (id_{A_1 \vee A_2} \mid (\beta_1 \mid \beta_2)) &= \text{(Lemma 31)} \\
(id_{A_1 \vee A_2} \cdot id_{A_1 \vee A_2}) \mid ((\alpha_1 \mid \alpha_2) \cdot (\beta_1 \mid \beta_2)) &= \text{(Lemma 31)} \\
(id_{A_1 \vee A_2} \cdot id_{A_1 \vee A_2}) \mid ((\alpha_1 \cdot \beta_1) \mid (\alpha_2 \cdot \beta_2)) &= \text{(comp. of identities)} \\
id_{A_1 \vee A_2} \mid ((\alpha_1 \cdot \beta_1) \mid (\alpha_2 \cdot \beta_2)) &= L_{AB}(\alpha \cdot \beta)
\end{aligned}
$$

third part *we show that the arrows 1 to 6 in the definition define the natural transformations that correspond to this definition.*

Given derivations $f = (f_1, f_2)$ in $\mathcal{A}$, $f_1 \colon A'_1[x \colon A_1]$ and $f_2 \colon A'_2[x \colon A_2]$, and $g \colon B[x \colon B']$ in $\mathcal{B}$, we have the following natural transformations (put $F = \vee$ and $G = \Delta$ in figure 1):

1. $Hom(\vee f, B) \circ L_{AB} \Rightarrow L_{A'B} \circ Hom(f, \Delta B)$: given a pair of derivations $\Sigma = (\Sigma_1 \colon B[x \colon A_1],\ \Sigma_2 \colon\ B[x \colon A_2])$,

$$\begin{pmatrix} A_1 & A_2 \\ \Sigma_1 & \Sigma_2 \\ B & B \end{pmatrix} \xrightarrow{L_{AB}} \dfrac{\begin{matrix} [A_1] & [A_2] \\ \Sigma_1 & \Sigma_2 \\ A_1 \vee A_2 & B \quad B \end{matrix}}{B} \xrightarrow{Hom(\vee f, B)} \dfrac{\dfrac{\begin{matrix} [A'_1] & [A'_2] \\ f_1 & f_2 \\ A_1 & A_2 \end{matrix}}{A'_1 \vee A'_2 \quad A_1 \vee A_2 \quad A_1 \vee A_2} \quad \begin{matrix} [A_1] & [A_2] \\ \Sigma_1 & \Sigma_2 \\ B & B \end{matrix}}{A_1 \vee A_2 \qquad\qquad B}$$

$$\begin{pmatrix} A_1 & A_2 \\ \Sigma_1 & \Sigma_2 \\ B & B \end{pmatrix} \xrightarrow{Hom(f, \Delta B)} \begin{pmatrix} [A'_1] & [A'_2] \\ f_1 & f_2 \\ A_1 & A_2 \\ \Sigma_1 & \Sigma_2 \\ B & B \end{pmatrix} \xrightarrow{L_{A'B}} \dfrac{\begin{matrix} [A'_1] & [A'_2] \\ f_1 & f_2 \\ A_1 & A_2 \\ \Sigma_1 & \Sigma_2 \end{matrix}}{A'_1 \vee A'_2 \quad B \quad B} \qquad B$$

and this natural transformation corresponds to the sequence ⟨MDP, ∨-reduction⟩.

2. $Hom(\vee A', g) \circ L_{A'B} \Rightarrow L_{A'B'} \circ Hom(A', \Delta g)$: given a pair of derivations $\Sigma = (\Sigma_1 \colon B[x \colon A'_1],\ \Sigma_2 \colon\ B[x \colon A'_2])$, $Hom(\vee A', g) \circ L_{A'B}(\Sigma)$ reduces to $L_{A'B'} \circ Hom(A', \wedge g)(\Sigma)$ and this natural transformation corresponds to MDP.

3. $K_{A'B} \circ Hom(\vee f, B) \Rightarrow Hom(f, \Delta B) \circ K_{AB}$: given a derivation $\Sigma \colon B[x \colon A_1 \vee A_2]$, $Hom(f, \Delta B) \circ K_{AB}(\Sigma)$ ∨-reduces to $K_{A'B} \circ Hom(\vee f, B)$ and this natural transformation corresponds to a pair of ∨-reductions.

4. $K_{A'B'} \circ Hom(\vee A', g) \Rightarrow Hom(A', \Delta g) \circ K_{A'B}$: given a derivation $\Sigma \colon B[x \colon A'_1 \vee A'_2]$, $K_{A'B'} \circ Hom(\vee A', g)(\Sigma) = Hom(A', \Delta g) \circ K_{A'B}$ and this natural transformation corresponds to the identity reduction.

5. $\alpha \colon id_{Hom_{Cat}(\vee A, B)} \Rightarrow L_{AB} \circ K_{AB}$: given a derivation $\Sigma \colon\ B[x \colon A_1 \vee A_2]$, $id_{Hom_{Cat}(\vee A, B)}(\Sigma)$ reduces to $L_{AB}K_{AB}(\Sigma)$ and this natural transformation corresponds to the sequence ⟨∨-expansion, MDP⟩

6. $\beta \colon K_{AB} \circ L_{AB} \Rightarrow id_{Hom_{Cat}(A, \Delta B)}$: given a pair of derivations $\Sigma = (\Sigma_1 \colon B[x \colon A_1], \Sigma_2 \colon B[x \colon A_2])$, $K_{AB}L_{AB}(\Sigma)$ reduces to $id_{Hom_{Cat}(\vee A, B)}(\Sigma)$ and this natural transformation corresponds to a pair of ∨-reductions.

fourth part *finally, we show that* $(\beta; id_{K_{AB}}) \cdot (id_{K_{AB}}; \alpha) = id_{K_{AB}}$ *and* $(id_{L_{AB}}; \beta) \cdot (\alpha; id_{L_{AB}}) = id_{L_{AB}}$ *hold.* The proof of these equalities is similar to the fourth part of the last section.

3.3 Implication

Implication is neither lax right 2-adjoint nor lax left 2-adjoint to conjunction but it has some of these two properties and we thought it would be interesting to show and discuss it here. We show what can be proved and point out the reasons why it is neither of the 2-adjunction forms. We also separate it in four parts:

first part *given the 2-category $\mathcal{PT}$, we show that $\rightarrow\colon \mathcal{PT} \rightarrow \mathcal{PT}$ and that $\wedge\colon \mathcal{PT} \rightarrow \mathcal{PT}$ are lax 2-functors.*

Lemma 3.31. *$\wedge\colon \mathcal{PT} \rightarrow \mathcal{PT}$ such that $\wedge(A) = A \wedge B$, $\wedge(f) = f \wedge B = \langle f \circ \pi_1, \pi_2 \rangle$ and $\wedge(\alpha) = \langle \alpha; id_{\pi_1}, id_{\pi_2} \rangle$ for every formula A, derivation f and rex α, is a lax 2-functor.*

Proof. This lemma is a corollary of lemma 3.12

Lemma 3.32 *$\rightarrow\colon \mathcal{PT} \rightarrow \mathcal{PT}$ such that $\rightarrow(B) = X \rightarrow B$, $\rightarrow(f) = X \rightarrow f =$*

$$\dfrac{\dfrac{X \rightarrow B \qquad [X]}{B}}{\dfrac{f}{\dfrac{B'}{X \rightarrow B'}}} = f \circ \widehat{eval\langle \rightarrow B, X \rangle}$$

where $\hat{g}$ is the canonical morphism from C to B^A such that g goes from $C \times A$ to B and $\rightarrow(\alpha) = id_r; (\alpha; id_s)$, where $r\colon B \rightarrow (X \rightarrow B)$ and $s\colon (X \rightarrow B, X) \rightarrow B$, for every formula B, derivation $f\colon B \rightarrow B'$ and rex $\alpha\colon f \Rightarrow f'$, is a lax 2-functor.

(iii) Given a formula A, both $\rightarrow(id_A)$ and $id_{\rightarrow(A)}$ are the derivation $X \rightarrow A$;
(iv) Given derivations $f\colon A \rightarrow B$ and $g\colon B \rightarrow C$, $\rightarrow(f) \circ \rightarrow(g)$reduces to $\rightarrow(f \circ g)$.
(vi) Given a derivation f, $\rightarrow(id_f) = id_r; id_f; id_s$ which is the identity reduction that goes from $\rightarrow(f)$ to $\rightarrow(f)$.
(vii) Given reductions $\alpha\colon f \Rightarrow g$ and $\beta\colon g \Rightarrow h$,

$$\begin{aligned}
\rightarrow(\alpha) \cdot \rightarrow(\beta) = \\
(id_r; (\alpha; id_s)) \cdot (id_r; (\beta; id_s)) &= \text{(2-comp. property)} \\
(id_r \cdot id_r); ((\alpha; id_s) \cdot (\beta; id_s)) &= \text{(2-comp. property)} \\
(id_r \cdot id_r); ((\alpha \cdot \beta); (id_s \cdot id_s)) &= \text{(comp. of identities)} \\
id_r; ((\alpha \cdot \beta); id_s) &= \rightarrow(\alpha \cdot \beta)
\end{aligned}$$

(viii) We only remark that applying $\rightarrow \alpha$ and $\rightarrow \beta$ followed by $\rightarrow$-reduction is the same as applying $\rightarrow$-reduction followed by $\rightarrow \alpha$ and $\rightarrow \beta$.

second part *we show that there exists an arrow K_{BC} from $Hom_{Cat}(C \wedge X, B)$ to $Hom_{Cat}(C, X \rightarrow B)$ and an arrow L_{BC} from $Hom_{Cat}(C, X \rightarrow B)$ to $Hom_{Cat}(C \wedge X, B)$.*

Lemma 3.33. *There exists a functor $K_{BC}\colon Hom_{Cat}(C \wedge X, B) \rightarrow Hom_{Cat}(C, X \rightarrow B)$.*

Define $K_{BC}(\Pi) = \dfrac{\dfrac{C \quad [X]}{C \wedge X}\ \Pi}{\dfrac{B}{X \to B}}$ for every Π in $Hom_{Cat}(C \wedge X, B)$ and $K_{BC}(\alpha) = id_r; \alpha; id_p$, where $p\colon (C, X) \to C \wedge X$, for every $\alpha \in Hom_{Cat}(C \wedge X, B)$. (iii) Given a derivation Π, $K_{BC}(id_(\Pi) = id_r; id_\Pi; id_p$ which is the identity reduction that goes from $K_{BC}(\Pi)$ to $K_{BC}(\Pi)$;
(iv) Given reductions α and β,

$$\begin{aligned} K_{BC}(\alpha) \cdot K_{BC}(\beta) = \\ (id_r; (\alpha; id_p)) \cdot (id_r; (\beta; id_p)) &= \text{(2-comp. property)} \\ (id_r \cdot id_r); ((\alpha; id_p) \cdot (\beta; id_p)) &= \text{(2-comp. property)} \\ (id_r \cdot id_r); ((\alpha \cdot \beta); (id_p \cdot id_p)) &= \text{(comp. of identities)} \\ id_r; ((\alpha \cdot \beta); id_p) &= K_{BC}(\alpha \cdot \beta) \end{aligned}$$

Lemma 3.34. *There exists a functor* $L_{BC}\colon Hom_{Cat}(C, X \to B) \to Hom_{Cat}(C \wedge X, B)$.

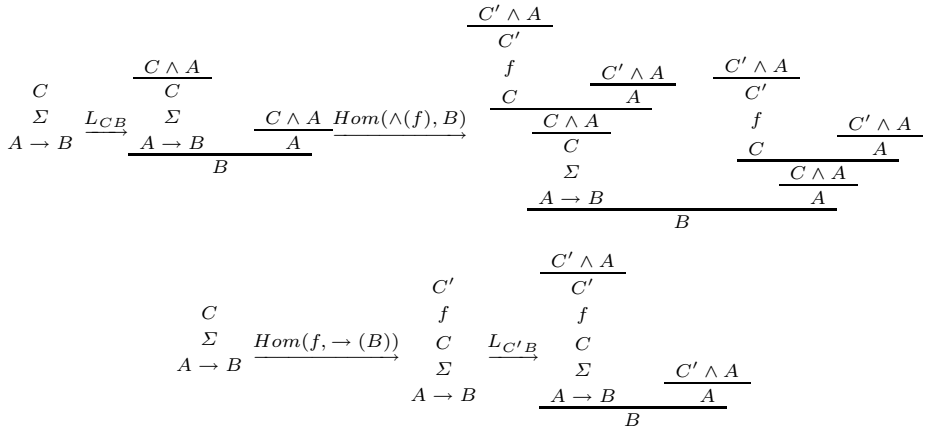
Put $L_{BC}(\Psi) = \dfrac{\dfrac{\dfrac{C \wedge X}{C}\ \Psi}{X \to B} \quad \dfrac{C \wedge X}{X}}{B}$ for every $\Psi \in Hom_{Cat}(C, X \to B)$ and $L_{BC}(\alpha) = (\alpha; id_{\pi_1}) \mid id_{\pi_2}$ for every $\alpha\colon \Pi \Rightarrow \Pi' \in Hom_{Cat}(C, X \to B)$. (iii) Given a derivation Π, $L_{BC}(id_\Pi) = (id_\Pi; id_{\pi_1}) \mid id_{\pi_2}$ which is the identity reduction that goes from $L_{BC}(\Pi)$ to $L_{BC}(\Pi)$;
(iv) Given reductions α and β,

$$\begin{aligned} L_{BC}(\alpha) \cdot L_{BC}(\beta) = \\ ((\alpha; id_{\pi_1}) \mid id_{\pi_2}) \cdot ((\beta; id_{\pi_1}) \mid id_{\pi_2}) &= \text{(Lemma 31)} \\ ((\alpha; id_{\pi_1}) \cdot (\beta; id_{\pi_1})) \mid (id_{\pi_2} \cdot id_{\pi_2}) &= \text{(2-comp. property)} \\ ((\alpha \cdot \beta); (id_{\pi_1} \cdot id_{\pi_1})) \mid (id_{\pi_2} \cdot id_{\pi_2}) &= \text{(comp. of identities)} \\ ((\alpha \cdot \beta); id_{\pi_1}) \mid id_{\pi_2} &= L_{BC}(\alpha \cdot \beta) \end{aligned}$$

third part *we show that the arrows 1 to 6 in the definition define natural transformations that correspond either to the lax right 2-adjunction definition or to the lax left 2-adjunction definition, i.e., we show that the natural transformations correspond to the items 1, 2 and 3 are in accordance to the lax left 2-adjunction definition, the item 4 is in accordance to the lax right 2-adjunction definition and that 5 and 6 is neither of the 2-adjunction forms.*

Given derivations $g\colon B'[x\colon B]$ and $f\colon C'[x\colon C]$ in $\mathcal{PT}$, we have the following natural transformations (sometimes we have $F = \to$ and $G = \wedge$ and sometimes we have $F = \wedge$ and $G = \to$ in figure 1):

1. $Hom(\wedge(f), B) \circ L_{CB} \Rightarrow L_{C'B} \circ Hom(f, \to (B))$ in accordance to lax left 2-adjunction: given a derivation $\Sigma\colon A \to B[x\colon C]$,



As $Hom(\wedge(f), B) \circ L_{CB}(\Sigma)$ $\wedge$-reduces to $L_{C'B} \circ Hom(f, \rightarrow (B))(\Sigma)$, this natural transformation corresponds to $\wedge$-red$|$ $\wedge$-red.

2. $Hom(\wedge(C'), g) \circ L_{C'B} \Rightarrow L_{C'B'} \circ Hom(C', \rightarrow (g))$ in accordance to lax left 2-adjunction: given a derivation $\Sigma\colon A \rightarrow B[x\colon C']$, $Hom(\wedge(C'), g) \circ L_{C'B}(\Sigma)$ $\rightarrow$-reduces to $L_{C'B'} \circ Hom(C', \rightarrow (g))(\Sigma)$ and this natural transformation corresponds to $\rightarrow$-reduction.

3. $K_{C'B} \circ Hom(\wedge(f), B) \Rightarrow Hom(f, \rightarrow B) \circ K_{CB}$ in accordance to lax left 2-adjunction: given a derivation $\Sigma\colon B[x\colon C \wedge A]$, $Hom(f, \rightarrow B) \circ K_{CB}(\Sigma)$ $\wedge$-reduces to $Hom(f, \rightarrow B) \circ K_{CB}(\Sigma)$ and this natural transformation corresponds to $(\wedge\text{-red}|\ \wedge\text{-red}); id_{\Sigma}; id_r$.

4. $Hom(C', \rightarrow (g)) \circ K_{C'B} \Rightarrow K_{C'B'} \circ Hom(\wedge(C'), g)$ in accordance to the definition of lax right 2-adjunction: given a derivation $\Sigma\colon B[x\colon C' \wedge A]$, $Hom(C', \rightarrow (g)) \circ K_{C'B}(\Sigma)$ $\rightarrow$-reduces to $K_{C'B'} \circ Hom(\wedge(C'), g)(\Sigma)$ and this natural transformation corresponds to $\rightarrow$-reduction.

5. Neither $id_{Hom_{Cat}(C \wedge A, B)} \Rightarrow L_{CB} \circ K_{CB}$ nor $L_{CB} \circ K_{CB} \Rightarrow id_{Hom_{Cat}(C \wedge A, B)}$: given a derivation $\Sigma\colon B[x\colon C \wedge A]$, and there is neither a rex from $id_{Hom_{Cat}(C \wedge A, B)}$ to $L_{CB} \circ K_{CB}$ nor from $L_{CB} \circ K_{CB}$ to $id_{Hom_{Cat}(C \wedge A, B)}$.

If we use α-reduction, that is, the reduction that corresponds to $\dfrac{\dfrac{A \wedge B}{A}\quad \dfrac{A \wedge B}{B}}{A \wedge B} \triangleright A \wedge B$, then there exists a rex from $L_{CB} \circ K_{CB}(\Sigma)$ to $id_{Hom_{Cat}(C \wedge A, B)}(\Sigma)$, which is the $\rightarrow$-red followed by this λ-calculus reduction and then $L_{CB} \circ K_{CB} \Rightarrow id_{Hom_{Cat}(C \wedge A, B)}$ is in accordance to lax left 2-adjunction.

6. Neither $K_{CB} \circ L_{CB} \Rightarrow id_{Hom_{Cat}(C, A \rightarrow B)}$ nor $id_{Hom_{Cat}(C, A \rightarrow B)} \Rightarrow K_{CB} \circ L_{CB}$: given a derivation $\Sigma\colon A \rightarrow B[x\colon C]$, there is neither a rex from $id_{Hom_{Cat}(C, A \rightarrow B)}$ to $K_{CB} \circ L_{CB}$ nor from $K_{CB} \circ L_{CB}$ to $id_{Hom_{Cat}(C, A \rightarrow B)}$.

If we use η-reduction that is, the reduction that corresponds to $\dfrac{A \rightarrow B \qquad A}{\dfrac{B}{A \rightarrow B}}$ $\triangleright A \rightarrow B$, then there exists a rex from $K_{CB} \circ L_{CB}(\Sigma)$ to $id_{Hom_{Cat}(C, A \rightarrow B)}(\Sigma)$,

which is the $\wedge$-red followed by this λ-calculus reduction and then $K_{CB} \circ L_{CB} \Rightarrow id_{Hom_{Cat}(C \wedge A,B)}$ is in accordance to lax right 2-adjunction.

fourth part *we show that neither of the equalities of the definitions of lax left and lax right 2-adjunction hold, i.e., we show that neither* $(\beta; id_{K_{CB}}) \cdot (id_{K_{CB}}; \alpha) = id_{K_{CB}}$, $(id_{L_{CB}}; \beta) \cdot (\alpha; id_{L_{CB}}) = id_{L_{CB}}$, $(id_{K_{CB}}; \alpha) \cdot (\beta; id_{K_{CB}}) = id_{K_{CB}}$ *nor* $(\alpha; id_{L_{CB}}) \cdot (id_{L_{CB}}; \beta) = id_{L_{CB}}$ *hold.*

Considering α and η-reduction, we would have the natural transformation α as in the definition of lax right 2-adjunction, i.e., $\alpha \colon L_{CB} \circ K_{CB} \Rightarrow id_{Hom_{Cat}(C \wedge A,B)}$, and β as in the definition of lax left 2-adjunction, i.e., $\beta \colon K_{CB} \circ L_{CB} \Rightarrow id_{Hom_{Cat}(C,A \to B)}$. Even though, we would not have neither of the equalities:

1. $\beta; id_{K_{CB}}$ goes from $(K_{CB} \circ L_{CB}) \circ K_{CB}$ to $id_{Hom_{Cat}(C,A \to B)} \circ K_{CB} = K_{CB}$ and $id_{K_{CB}}; \alpha$ goes from $K_{CB} \circ (L_{CB} \circ K_{CB})$ to $K_{CB} \circ id_{Hom_{Cat}(C \wedge A,B)} = K_{CB}$, therefore there is not a composition of $\beta; id_{K_{CB}}$ with $id_{K_{CB}}; \alpha$.

2. $\alpha; id_{L_{CB}}$ goes from $(L_{CB} \circ K_{CB}) \circ L_{CB}$ to $id_{Hom_{Cat}(C \wedge A,B)} \circ L_{CB} = L_{CB}$ and $id_{L_{CB}}; \beta$ goes from $L_{CB} \circ (K_{CB} \circ L_{CB})$ to $L_{CB} \circ id_{Hom_{Cat}(C,A \to B)} = L_{CB}$, therefore there is not a composition of $\alpha; id_{L_{CB}}$ with $id_{L_{CB}}; \beta$.

4 On the Interpretation for $\bot$

We note here how cumbersome is the categorical interpretation of the $\bot$. Joyal [2] (p.116) has observed that in a cartesian closed category $\mathcal{C}$ with initial object $\bot$ and an object A, there exists only one arrow from A to $\bot$. Such a result is not supported by proof theoretical means. For example, the derivation $\dfrac{\begin{matrix}[A] & [A] \\ f & g \\ \bot & \bot\end{matrix}}{\dfrac{\bot \wedge \bot}{\bot}\,\pi}$ reduces to $\begin{matrix}[A] \\ f \\ \bot\end{matrix}$ if π is the rule $\dfrac{A \wedge B}{A}$ and to $\begin{matrix}[A] \\ g \\ \bot\end{matrix}$ if π is the rule $\dfrac{A \wedge B}{B}$ and, as there is no rex between f and g, we come to no conclusion about a relation between f and g.

The constant $\bot$ cannot be seen as a 2-initial object for, given derivations Π and Ψ from $\bot$ to a formula A, it is not the case that there always exists a rex from Π to Ψ that is an isomorphism because, for example, the derivation $\dfrac{\dfrac{\bot}{\bot \wedge A}}{\bot}$ reduces to $\bot$ but $\bot$ does not expands to $\dfrac{\dfrac{\bot}{\bot \wedge A}}{\bot}$.

For a similar reason, $\bot$ cannot be seen as a lax 2-initial object. For example, there is no rex between $\dfrac{\dfrac{\bot}{\bot \wedge A}}{\bot}$ and $\dfrac{\dfrac{\bot}{A} \qquad \dfrac{\bot}{A \to \bot}}{\bot}$. Note that both these derivations reduce to $\bot$ and therefore, they are 1-categorically represented by the same arrow, viz., the identity arrow from $\bot$ to $\bot$.

A possible proof-theoretical solution is to consider structural reductions (global reductions) in our 2-categorical approach. According to Prawitz's definition, both the derivations

$$\dfrac{\dfrac{\dfrac{C \wedge \neg C}{\bot}}{A}\,1 \qquad \dfrac{D \wedge (A \to B)}{A \to B}}{B} \qquad \dfrac{\dfrac{A \wedge (B \to A)}{B \to A} \qquad \dfrac{\dfrac{A \wedge (A \to B)}{A \to B} \qquad \dfrac{A \wedge (B \to A)}{A}}{B}}{A} \qquad (\star)$$

are normal. However, these derivations are redundant: on the derivation on the left side, as any formula can be deduced from $\bot$, the conclusion of the derivation could have been deduced in (1) and on the one on the right side there is already a derivation of the conclusion as sub-deduction.

Thus, to these kinds of derivations we need, instead of local reductions that deal with introduction and elimination of logical operators, structural reductions, which work on a global level, i.e., as the name indicates it, on the structure of the derivation.

$$\dfrac{\dfrac{\begin{array}{c}\Pi_1\\ \bot\\ \hline A\end{array} \qquad \begin{array}{c}\Pi_2\\ A \to B\end{array}}{B} \rhd_{P-H} \dfrac{\begin{array}{c}\Pi_1\\ \bot\end{array}}{B} \qquad\qquad \begin{array}{c}\Gamma\\ \Pi_1\\ A\\ \Pi_2\\ A\end{array} \rhd_E \begin{array}{c}\Gamma'\\ \Pi_1\\ A\end{array}$$

The reduction on the left is a structural reduction defined in [4] in order to prove that all derivations from a formula A to $\bot$ are equivalent, providing a proof-theoretical argument to Joyal's observation. Therefore, we believe that the inclusion of this reduction approximates semantically Category Theory to Proof Theory.

In [1], Ekman defines the reduction on the right side, where Γ and Γ' are sets of premises and $\Gamma' \subseteq \Gamma$. With this new definition, the derivation $(\star)$ can be reduced to $\dfrac{A \wedge (B \to A)}{A}$.

Ekman's reduction cannot be seen 2-categorically. When we begin examining the 2-categorical commutativity, the property (1) in section 1 does not hold, e.g.,

$$\dfrac{A \qquad \dfrac{\begin{array}{c}[A]\\ \Pi\\ B\end{array}}{A \to B}}{B} \rhd_E \begin{array}{c}A\\ \Pi\\ B\end{array} \quad \text{but} \quad \dfrac{\dfrac{\begin{array}{c}A\\ \Pi'\\ C\end{array}}{A} \qquad \dfrac{\begin{array}{c}[A]\\ \Pi\\ B\end{array}}{A \to B}}{B} \not\rhd_E \begin{array}{c}A\\ \Pi'\\ C\\ A\\ \Pi\\ B\end{array}$$

meaning that there is not a correspondence between 2-categorical and proof-theoretical composition. In other words this means that there exists 1-cells $f, g\colon A \to B$ with a 2-cell $\alpha\colon f \Rightarrow g$ and 1-cells $f', g'\colon A \to A$ with a 2-cell $\beta\colon f' \Rightarrow g'$ such that there is no 2-cell $\gamma = \alpha; \beta$ from $f \circ f'$ to $g \circ g'$.

5 Conclusion

We adopted the 2-Categorical view of proofs in order to categorically focus on Prawitz identity criteria on derivations. In fact, the main proof-theoretical

meanings on derivations are adequately lifted to this 2-Categorical framework. We detailed the lax-adjunctions that provide meaning to the intuitionistic logical constants as predicted by Seely. Unfortunately, implication is not a 2-Categorical adjunction, unless $Hom_{Cat}(X, Y)$ is taken as a preorder category. The $\perp$ is not a 2-Categorical notion either. Even considering (unusual) structural reductions, we cannot mend this situation. We would like to point out that the conclusions drawn by this article might be compared to what we do know about Prawitz conjecture in terms of derivations and formulas as they are taken in the usual mathematical and logical discourse.

References

1. Ekman, J.: Normal Proofs in Set Theory. PhD thesis, University of Goteborg (1994)
2. Lambek, J., Scott, P.J.: Introduction to Higher Order Categorial Logic, 1st edn., Cambridge. Cambridge Studies in Advanced Mathematics, vol. 7 (1986)
3. Mann, C.R.: Connections between Proof Theory and Category Theory. PhD thesis, Oxford (1973)
4. Pereira, L.C., Haeusler, H.: Structured reductions and the identity problem. In: Unilog (2007)
5. Prawitz, D.: Ideas and results in proof theory. In: Fenstad, J. (ed.) Proc. 2nd Scandinavian Logic Symposium, pp. 237–309. North-Holland, Amsterdam (1971)
6. Prawitz, D.: Natural Deduction, A Proof-Theoretical Study. Dover, New York (1965)
7. Seely, R.A.G.: Weak adjointness in proof theory. In: Applications of sheaves (Proc. Res. Sympos. Appl. Sheaf Theory to Logic, Algebra and Anal., Univ. Durham, Durham, 1977). Lecture Notes in Math., vol. 753, pp. 697–701. Springer, Berlin (1979)

WOMM: A Weak Operational Memory Model

Arnab De[1], Abhik Roychoudhury[2], and Deepak D'Souza[1]

[1] Department of Computer Science and Automation,
Indian Institute of Science, Bangalore, India
{arnabde,deepakd}@csa.iisc.ernet.in
[2] School of Computing, National University of Singapore, Singapore
abhik@comp.nus.edu.sg

Abstract. Memory models of shared memory concurrent programs define the values a read of a shared memory location is allowed to see. Such memory models are typically weaker than the intuitive sequential consistency semantics to allow efficient execution. In this paper, we present WOMM (abbreviation for Weak Operational Memory Model) that formally unifies two sources of weak behavior in hardware memory models: reordering of instructions and weakly consistent memory. We show that a large number of optimizations are allowed by WOMM. We also show that WOMM is weaker than a number of hardware memory models. Consequently, if a program behaves correctly under WOMM, it will be correct with respect to those hardware memory models. Hence, WOMM can be used as a formally specified abstraction of the hardware memory models. Moreover, unlike most weak memory models, WOMM is described using operational semantics, making it easy to integrate into a model checker for concurrent programs. We further show that WOMM has an important property - it has sequential consistency semantics for datarace-free programs.

1 Introduction

With the push towards multicore platforms in the coming years, the importance of concurrent programming is steadily growing. Increasingly, it is felt that computer architects will be moving towards processors with many cores and concurrent/parallel programming will become much more mainstream.

In general, programmers tend to find parallel/concurrent programming harder than sequential programming, owing to the many possible executions of a program for any given input. Shared-memory concurrent programming languages describe the semantics of concurrency via a *Memory Model* — it describes which writes can be visible to a program read operation. Programmers intuitively consider the memory model of a concurrent program as Sequential Consistency (SC) [1], where each process proceeds in program order and the operations from each process are interleaved and a read sees the last write to the same memory location in that interleaving. Although the SC semantics is easy for the programmers to understand, it does not allow many compiler and hardware optimizations.

T. Margaria and B. Steffen (Eds.): ISoLA 2010, Part I, LNCS 6415, pp. 519–534, 2010.

As a simple example, one may consider the program fragment in Figure 1. Throughout the paper, we use x, y, ... to denote shared memory locations and r1, r2, ... to denote local variables/ registers.

Initially, x == y == 0

Proc 1:	Proc 2:
x = 1;	r1 = y;
y = 1;	r2 = x;

r1 == 1, r2 == 0 not allowed by the SC

Fig. 1. Not allowed by SC - I

Initially, x == 0

Proc 1:	Proc 2:	Proc 3:
x = 1;	r1 = x;	r3 = x;
x = 2;	x = 3;	x = 4;
	r2 = x;	r4 = x;

r1 == r4 == 1, r2 == r3 == 2 not allowed by SC

Fig. 2. Not allowed by SC - II

In this example, we would normally assume r1 $\leq$ r2 at the end of the program. In other words, SC does not allow r1 == 1 $\wedge$ r2 == 0 since y is set after x in Proc 1. However, Proc 1 may reorder the writes at runtime (as evidenced by hardware multiprocessor memory models such as Sparc Partial Store Order (PSO) [2]), thereby making the result r1 == 1 $\wedge$ r2 == 0 possible at the end of the program.

Similarly, the behavior demonstrated in the example from Figure 2 cannot be produced by an SC execution. Nevertheless, the interconnection network may cause Proc 3 to see the writes from Proc 1 out of order, but Proc 2 may see them in the program order, causing this behavior.

To accommodate such optimizations, weaker or more relaxed memory models have been proposed. Such memory models typically allow more behaviors than the SC. Such intricacies in the memory model makes understanding concurrent program behavior particularly difficult.

Conventionally *hardware multiprocessor memory models* — such as Total Store Order (TSO), Partial Store Order (PSO) and Relaxed Memory Order (RMO) of Sun Sparc [3,2] have been specified using a *reordering table*, denoting whether two instructions accessing two different memory locations can be reordered at runtime. Similarly, the official memory models for IA-32 and AMD-64 architectures are defined as a set of informal rules [4,5]. Recently, a rigorous, axiomatic memory model [6] has been proposed for x86 multiprocessors that matches the informal documentations. These memory models differ from each other in subtle but complex ways. Moreover, the specifications are either informal or axiomatically defined, making them difficult to follow. As a result, a compiler writer or a low-level programmer may not get an abstract view of the underlying hardware memory models.

We have observed that weak behavior in multiprocessors is mostly caused by two features. A processor can reorder instructions accessing different memory locations (as in Figure 1) or writes to the same memory location may be visible to different processors in different orders (as in Figure 2). In this paper, we present formal semantics of a Weak Operational Memory Model, referred as WOMM, that combines these two features. We use a language that contains some of

the essential features of multiprocessor programs. We use a simple operational semantics to specify our memory model, making it easy to follow for the compiler writers and low-level programmers. The operational style of our specification allows easy integration to program analysis tools such as model checkers. We demonstrate that a number of optimizations are allowed by WOMM. We show that WOMM is weaker than the Location Consistency (LC) [7] memory model, which, in turn, makes it weaker than a number of memory models such as TSO, PSO, RMO etc. Consequently, any execution allowed by these hardware memory models will be allowed by WOMM. As a result, if a compiler assumes WOMM as the underlying hardware memory model, the compilation will be correct for all these hardware memory models. Finally, we show that WOMM has an important property — it guarantees sequential consistency for datarace-free programs.

2 Program Model

We consider a simple and abstract program model for the sake of presentation. A multiprocessor system consists of a finite number of processors, each with an unique id. All the processors have a finite number of local registers. Memory is shared among all the processors.

The program consists of one executable per processor. Each executable has a finite number of static instructions, with designated first and last instructions. In the rest of the paper, we identify the executables by the processors they run on. Instructions can be categorized into following types. We use the following conventions: `r`, `r1`, ... represent registers, `[r]` a shared memory location whose address is equal to the contents of `r`, *exp* a local expression and `L` a program label.

Read (Rd) (`r1 = [r2]`): Reads the content of the memory location `[r2]` to the register `r1`.

Write (Wr) (`[r1] = r2`): Writes the value of the register `r2` to the memory location `[r1]`.

Local computation (Loc) (`r =` *exp*): Assigns the value of the expression *exp*, which does not have any memory access, to the register `r`.

Conditional goto (Cond) (`if r goto L`): If the register `r` has a non-zero value, program counter is set to the `L`, otherwise to the next program location.

Unconditional goto (Jmp) (`goto L`): Sets program counter to the `L`.

Acquire (Acq) (`acquire [r]`): Acquires the memory location `[r]`. Once a processor executes acquire on a memory location, no other processor can acquire that location till the former processor *releases* it. Moreover, it works as a one-way barrier — instructions after an acquire cannot be reordered before it at runtime.

Release (Rel) (`release [r]`): Releases the memory location `[r]` enabling other processors to acquire it. It also works as a one-way barrier — instructions before a release cannot be reordered after it at runtime.

A *synchronization instruction* is either an acquire or a release instruction. Note that a large number of synchronization primitives found in real hardwares can be closely modeled with these two types of instructions. The acquire and release instructions are close to the load-acquire/ store-release instructions of the Itanium architecture [8]. A two-way memory barrier instruction can be modeled by an acquire instruction immediately followed by a release instruction on the same memory location, where there is one memory location per processor designated for this purpose. An atomic operation involving access to a single memory location (such as test-and-set and compare-and-exchange) can be modeled as acquire instruction on that memory location, followed by instructions implementing the atomic instruction, followed by a release instruction on the same memory location (assuming that memory location is not accessed in any other way).

In the rest of the paper, short names of the instructions in the above list represent the set of all possible instructions of the corresponding type and the representative form given along is used to represent an arbitrary instruction of that type. The word *instruction* may refer to a static instruction or a dynamic, runtime instance of a static instruction. depending on the context. We also use the term *action* to refer to dynamic instructions.

3 Operational Semantics

In this section we formally describe the operational semantics of *WOMM*, a weak abstract memory model for shared memory multiprocessors. Informally, instructions are *issued* in program order and put into an *instruction queue*. An instruction can *commit* any time after all the instructions it depends on commit. Each memory location, instead of containing a single value, contains the set of all accesses to that location along with some *causal order* among them. A read can return any value from this list under certain consistency restrictions.

3.1 Structure of States

Given a program P, a program state Ω consists of a *global state* Θ and one *local state* Φ_p for each processor p.

A global state Θ consists of the following components:

- A map Π from memory locations to *memory states*. Informally, a memory state in our semantics contains all the accesses to that memory location along with the *causal orders* among them. Formal description of a memory state is given below.
- A map L from memory locations to *lock states*. A lock state may contain either a processor id, denoting the location is acquired by the processor with id, or 0, denoting the location is free.

A *memory state* is a *partially ordered multiset* (*pomset*), denoted by $(S, \preceq)$, where S is the multiset and $\preceq$ is the partial order. A multiset S is a collection

of *pomset items*. Being a multiset, it allows duplication of elements. Each item is added by some instruction. The pomset items can be of following types. Here *val* denotes a value, *pid* denotes a the processor id performing the corresponding write/ release/ acquire action and *loc* denotes a memory location.

1. A *write item* (WI) which is a $\langle val, pid \rangle$ pair.
2. A *read item* (DI) which is simply $\langle pid \rangle$.
3. A *release item* (RI) which is a $\langle loc, pid \rangle$ pair.
4. An *acquire item* (AI) which is again a $\langle loc, pid \rangle$ pair.

The relation $\preceq$ is a partial order, i.e. a reflexive, antisymmetric and transitive binary relation on S. The corresponding irreflexive relation is denoted by $\prec$. Whenever this relation is updated (e.g. in Figure 3 and Figure 6), we assume that the new relation is reflexively and transitively closed to maintain the partial order property.

As the same local register name can be used in different unrelated instructions in a processor, it may create false dependencies among those instructions. For example, if two consecutive instructions assign to the same local register, there may not be any actual data flow between them, but they still cannot be reordered naïvely as a future read of that register may get affected by the reordering. To avoid such false dependency and facilitate reordering of instructions, local registers are renamed at runtime. A single local register can be associated with different dynamic registers at different points of the execution and each dynamic register can have different values. We assume that there are infinite number of dynamic registers available.

A local state Φ_p consists of the following components:

- A map μ_p that maps static register names to their current dynamic names.
- A map σ_p that maps dynamic registers and the program counter to the values they contain. The domain of values includes a special value `undef`.
- An instruction queue Q_p, which contains dynamic instructions (actions) that have been issued but not yet committed (see Section 3.2). Actions are ordered by the order of their issue in the queue. Registers are renamed with dynamic names for the actions residing in the queue (see Section 3.4).
- A map δ_p that maps an issued action to the pomset item corresponding to the action if the instruction is a read or a write or a synchronization instruction.

Initial State: In the initial state, all the maps in the local states of the processors and the instruction queue are empty (i.e. returns `undef` for any input) except the program counter which points to the first instructions of every executable. All memory locations are free. All memory states contain a special write element, containing an arbitrary initial value of the location, denoted by $WI\langle val, \top \rangle$.

3.2 Execution

An execution of a program is a total order of *operations* where each operation is an *issue* or a *commit* of an action. An operation can occur if all its premises

are met and causes some state transition. The premises and effects of operations are described in Section 3.4 and Section 3.5. The total order among operations in an execution is referred as the *occurs-before* relation (denoted by $\stackrel{ob}{\rightarrow}$).

If i is an action, $issue(i)$ and $commit(i)$ denote the corresponding issue and commit operations. Similarly, if op is an operation, $inst(op)$ denotes the corresponding dynamic instruction. The state transition from Ω to Ω' caused by an operation op from processor p is denoted by $\langle p : op, \Omega \rangle \rightarrow \Omega'$.

Execution of a program starts with issue of the first instruction from the any arbitrary processor. For each processor, the execution starts with the issue of the first instruction from that processor. A processor *finishes execution* when the last instruction, along with all previously issued instructions, of that processor is committed. The program *finishes execution* when all processors finish execution.

A *trace* is a sequence of operations, denoted as $\langle op_1, op_2, ... \rangle$. As occurs-before is a total order, any execution in WOMM can be expressed as a trace in a natural way. An execution represented as a trace is referred to as a *trace execution*.

3.3 Complete Execution and Observable State

We call an execution *complete* if all issued instructions are committed. Note that a complete execution might not be a finished execution.

The ultimate goal of our semantics is to define, given a closed program, the set of *observable states* reachable from an initial state, via a complete execution. An *observable state* is the value of the local registers and program counters of each processor. If p is a processor and `r` is register of p, the *value* of `r` is given by $\sigma_p(\mu_p(\mathtt{r}))$.

Note that an implementation obeying WOMM need not execute the instructions from different processors in a total order, but the observable state after executing a set of instructions by the implementation must be reachable via a complete trace execution of exactly the same set of instructions.

3.4 Semantics of Issue

Issues of instructions must be done in *program order*. As a consequence, the issue operations update the program counters. For the sake of simplicity, we do not show this requirement and effect in the formal rules. Issue of any instruction by a processor p also has the following effects:

- Renames the registers. If a register `r` is read in the instruction, it is renamed to $\mu_p(\mathtt{r})$. If it is assigned to, it is renamed with a new dynamic name and μ_p is updated to reflect that and $\sigma_p(\mu_p(r))$ is set to `undef`. It is important to follow the order as the same static register can be both read and assigned in a single instruction. This operation is denoted as *Rename*.
- Appends the renamed instruction to the instruction queue Q_p.

If there is an uncommitted *Cond* instruction issued from a processor, the next instruction to be issued is undefined. Hence an instruction can be issued by a

processor only if there is no *Cond* instruction in the corresponding instruction queue. Similarly, if the value of a dynamic register is `undef`, a read or write instruction dereferencing that register cannot be issued.

If the instruction is a read/write instruction, then the corresponding read/write item is added to the pomset corresponding to the memory location involved in that instruction. The write items do not have the value defined.If the instruction is a synchronization instruction, then the corresponding release/acquire item is added to all the pomsets. To mark that the synchronization instruction has not committed, the memory location in the corresponding release/acquire item is kept undefined. The newly added items are ordered after the existing items from the same processor. The operation of adding an item I to a pomset P during issue is denoted by *IssueUpdate* and is formally defined in Figure 3, where *Pid* maps an item to the corresponding processor. The map δ_p is updated accordingly.

$$
\begin{aligned}
&\mathit{IssueUpdate} \equiv \lambda I.\lambda P.(S', \preceq') \\
&\text{where} \\
&\qquad P = (S, \preceq) \\
&\qquad S' = S \cup \{I\} \\
&\qquad \preceq' = \preceq \cup \{(e, I) \mid e \in S \\
&\qquad\qquad \wedge(\mathit{Pid}(e) = \mathit{Pid}(I) \vee \mathit{Pid}(e) = \top)\}
\end{aligned}
$$

Fig. 3. Issue Update

Figure 4 shows the formal rules for issue by an arbitrary processor p. *Issue-Rd*, *Issue-Wr*, *Issue-Acq*, *Issue-Rel* and *Issue-Gen* are the rules for issue of read, write, acquire, release and other types of instructions, respectively. $\Omega[C_1; \ldots; C_n]$ denotes a state which is equal to Ω except the changes $C_1, \ldots, C_n$, in that order. Any change C_i can have two forms: $M|M'$ denotes the state component M is modified to M' and if M is a map, $M\{x \rightarrow v\}$ denotes that $M(x)$ is updated with the value v. In the quantifiers, x ranges over the set of memory locations. If l is the address of a memory location, $[l]$ denotes the corresponding memory location. The list append operation is denoted by ".".

3.5 Semantics of Commit

Actions in the queue can be committed out-of-order, as long as it follows the following ordering restrictions:

- Synchronization actions must be committed in program order.
- Any action must be committed after any local acquire action issued before it.
- Any action must be committed before any local release action issued after it.
- Any read action can commit only if there is no local uncommitted write to the same memory location issued before it.
- Any action reading a register can be committed only if the value of all registers read in the action are not `undef`.

(Issue-Rd)

$$\frac{\begin{array}{c} inst \equiv \texttt{r1} = [\texttt{r2}] \quad Cond \cap Q_p = \emptyset \quad I = DI\langle p \rangle \\ l = \sigma_p(\mu_p(\texttt{r2})) \neq \texttt{undef} \quad P = IssueUpdate(I, \Pi([l])) \end{array}}{\langle p : issue(inst), \Omega \rangle \rightarrow \Omega[Q_p | Q_p.Rename(inst);\ \Pi\{[l] \rightarrow P\};\ \delta_p\{inst \rightarrow I\}]}$$

(Issue-Wr)

$$\frac{\begin{array}{c} inst \equiv [\texttt{r1}] = \texttt{r2} \quad Cond \cap Q_p = \emptyset \quad I = WI\langle *, p \rangle \\ l = \sigma_p(\mu_p(\texttt{r2})) \neq \texttt{undef} \quad P = IssueUpdate(I, \Pi([l])) \end{array}}{\langle p : issue(inst), \Omega \rangle \rightarrow \Omega[Q_p | Q_p.Rename(inst);\ \Pi\{[l] \rightarrow P\};\ \delta_p\{inst \rightarrow I\}]}$$

(Issue-Acq)

$$\frac{\begin{array}{c} inst \equiv \texttt{acquire}[\texttt{r}] \quad Cond \cap Q_p = \emptyset \quad I = AI\langle *, p \rangle \\ \Pi' = \forall x : \Pi\{x \rightarrow IssueUpdate(I, \Pi(x))\} \end{array}}{\langle p : issue(inst), \Omega \rangle \rightarrow \Omega[Q_p | Q_p.Rename(inst);\ \Pi | \Pi';\ \delta_p\{inst \rightarrow I\}]}$$

(Issue-Rel)

$$\frac{\begin{array}{c} inst \equiv \texttt{release}[\texttt{r}] \quad Cond \cap Q_p = \emptyset \quad I = RI\langle *, p \rangle \\ \Pi' = \forall x : \Pi\{x \rightarrow IssueUpdate(I, \Pi(x))\} \end{array}}{\langle p : issue(inst), \Omega \rangle \rightarrow \Omega[Q_p | Q_p.Rename(inst);\ \Pi | \Pi';\ \delta_p\{inst \rightarrow I\}]}$$

(Issue-Gen)

$$\frac{inst \notin Rd \cup Wr \cup Sync \quad Cond \cap Q_p = \emptyset}{\langle t : issue(inst), \Omega \rangle \rightarrow \Omega[Q_p | Q_p.Rename(inst)]}$$

Fig. 4. Rules for Issue of an Instruction

– An acquire action can commit only if the corresponding memory location is not acquired or acquired by the same processor. A release action can commit if the memory location is already acquired by the same processor.

Informally, the effects of a commit on the program state are as follows. Here p refers to the processor performing the operation and *inst* refers to the dynamic instruction corresponding to the operation. Note that the registers mentioned here are the dynamic ones, already renamed during issue.

Read (`r1 = [r2]`): Updates $\sigma_p(\texttt{r1})$ with the value read. The value read can be any value from the set of write items returned by the function *ConsistentRead*, defined in Figure 5, where x is the memory location read, I is the corresponding read item (given by $\delta_p(inst)$) and *Val* returns the value associated with a write item. This function returns the set of complete write items such that there is no intervening write item between any returned write item and the read item in the pomset order and any returned write item is not ordered after the read item. Note that the relation $\preceq$ is closed transitively after each update (see Section 3.1). The read cannot commit if the returned set is empty.

$$
\begin{aligned}
&\mathit{ConsistentRead} \equiv \\
&\lambda x.\lambda I.\{e \mid \Pi(x) = (S, \preceq) \\
&\qquad \wedge\, e \in S \wedge\ e \in WI \\
&\qquad \wedge\, \nexists e' \in WI : e \prec e' \prec I \\
&\qquad \wedge\, \neg(I \prec e) \\
&\qquad \wedge\ \mathit{Val}(e) \neq *\}
\end{aligned}
$$

Fig. 5. Consistent Read

$$
\begin{aligned}
&\mathit{CommitUpdate} \equiv \\
&\lambda I.\lambda P.(S', \preceq') \text{ where} \\
&\quad P\ = (S, \preceq) \\
&\quad S' = S \\
&\quad \preceq' = \preceq \cup \{(e, I) \mid e \in S \\
&\qquad\qquad\qquad \wedge e \in RI \\
&\qquad\qquad\qquad \wedge \mathit{MemLoc}(e) = \mathit{MemLoc}(I)\}
\end{aligned}
$$

Fig. 6. Commit Update

Write (`[r1] = r2`): Completes the corresponding write item (given by $\delta_p(inst)$) with the value of the local register (given by $\sigma_p(\mathtt{r2})$).

Local computation (`r = exp`): Evaluates the local expression on RHS using the values given by σ_p and updates $\sigma_p(\mathtt{r})$. The evaluation of the expression is denoted by *Evaluate* function. If any of the local registers on RHS has value `undef`, *Evaluate* returns `undef`. Otherwise it follows the natural semantics of expression evaluation.

Acquire (`acquire [r]`): Completes the corresponding acquire item (given by $\delta_p(inst)$) with $[\sigma_p(\mathtt{r})]$.[1] Also modifies the lock state of the memory location $[\sigma_p(\mathtt{r})]$.

Release (`release [r]`): Completes the corresponding release item (given by $\delta_p(inst)$) with $[\sigma_p(\mathtt{r})]$. Also modifies the lock state of the memory location $[\sigma_p(\mathtt{r})]$.

Commit of any acquire action also updates the partial order of memory states. It makes the corresponding acquire item ordered after all previously committed release items on the same memory location. This operation, referred as *CommitUpdate*, is defined in Figure 6, where I is the acquire item, P is the pomset and *MemLoc* maps an acquire/release item to the memory location associated with it.

Cond and *Jmp* actions do not change the state, except removing themselves from the instruction queue.

Figures 7 show formal rules for committing an action. Note that the registers in these instructions are already renamed to dynamic names. We do not show the obvious requirement that the committing instruction must be present in the instruction queue and after it is committed, it is removed from the queue. $\mathit{Pre}(q, i)$ denotes the set of actions in instruction queue q which were issued before the action i and $\mathit{Post}(q, i)$ denotes the set of actions in instruction queue q issued after i, where $i \in q$. $\mathit{Wr}(\mathtt{x})$ denotes the set of write actions to the memory location `x`. We follow all the conventions defined in Section 3.4.

[1] Note that the update of pomset item by commit of a synchronization instruction affects all pomsets as the synchronization item is added to all the pomsets during issue.

(Commit-Rd)

$$\frac{\begin{array}{c} inst \equiv \texttt{r1} = [\texttt{r2}] \quad l = \sigma_p(\texttt{r2}) \quad WI\langle v, p'\rangle \in ConsistentRead([l], \delta_p(inst))) \\ Pre(Q_p, inst) \cap Acq = \emptyset \quad Pre(Q_p, inst) \cap Wr([l]) = \emptyset \end{array}}{\langle p\text{: } commit(inst), \Omega\rangle \rightarrow \Omega[\sigma_p\{\texttt{r1} \rightarrow v\}]}$$

(Commit-Wr)

$$\frac{inst \equiv [\texttt{r1}] = \texttt{r2} \quad l = \sigma_p(\texttt{r1}) \quad v = \sigma_p(\texttt{r2}) \neq \texttt{undef} \quad Pre(Q_p, inst) \cap Acq = \emptyset}{\langle p\text{: } commit(inst), \Omega\rangle \rightarrow \Omega[\delta_p\{[l] \rightarrow WI\langle v, p\rangle\}]}$$

(Commit-Loc)

$$\frac{inst \equiv \texttt{r} = exp \quad v = Evaluate(\sigma_p, exp) \neq \texttt{undef} \quad Pre(Q_p, inst) \cap Acq = \emptyset}{\langle p\text{: } commit(inst), \Omega\rangle \rightarrow \Omega[\sigma_p\{\texttt{r} \rightarrow v\}]}$$

(Commit-Acq)

$$\frac{\begin{array}{c} inst \equiv \texttt{acquire [r]} \quad l = \sigma_p(\texttt{r}) \quad (L([l]) = \langle 0\rangle \ \ or \ \ L([l]) = \langle p\rangle) \\ L' = L\{[l] \rightarrow \langle p\rangle\} \\ \delta'_p = \delta_p\{inst \rightarrow AI\langle [l], p\rangle\} \\ \Pi' = \forall x : \Pi\{x \rightarrow CommitUpdate(\delta'_p(inst), \Pi(x))\} \\ Pre(Q_p, inst) \cap Sync = \emptyset \end{array}}{\langle p\text{: } commit(inst), \Omega\rangle \rightarrow \Omega[L|L';\ \delta_p|\delta'_p;\ \Pi|\Pi']}$$

(Commit-Rel)

$$\frac{\begin{array}{c} inst \equiv \texttt{release [r]} \quad l = \sigma_p(\texttt{r}) \quad L([l]) = \langle p\rangle \\ L' = L\{[l] \rightarrow \langle 0\rangle\} \\ \delta'_p = \delta_p\{inst \rightarrow RI\langle [l], p\rangle\} \\ Pre(Q_p, inst) = \emptyset \end{array}}{\langle p\text{: } commit(inst), \Omega\rangle \rightarrow \Omega[L|L';\ \delta_p|\delta'_p]}$$

Fig. 7. Rules for Committing an Instruction

3.6 Abstract Execution

Although the trace execution semantics (Section 3.2) helps in generating and traversing executions of a program, it is sometimes useful to view the executions as *abstract executions*, in the style of [9], where the actions are related by few causal orders. In this section we show how a complete WOMM trace execution can be naturally translated to an abstract execution.

More specifically, an abstract execution is a tuple $\langle P, A, \xrightarrow{po}, \xrightarrow{so}, W, V, \xrightarrow{sw}, \xrightarrow{hb}\rangle$, where P is the program, A is the set of actions executed, $\xrightarrow{po}$ is the program order, $\xrightarrow{so}$ is the synchronization order, W is the write seen function, V is the value written function, $\xrightarrow{sw}$ is the synchronizes-with relation and $\xrightarrow{hb}$ is the happens-before relation. We define each component of the tuple in context of a complete WOMM execution as below:

- P is the closed program.
- A is the set of actions issued and committed. Note that in a complete execution, all issued actions are committed.

- Given two actions i and i', $i \xrightarrow{po} i'$ iff $issue(i) \xrightarrow{ob} issue(i')$ and $Pid(i) = Pid(i')$.
- Given two synchronization actions i and i', $i \xrightarrow{so} i'$ iff $commit(i) \xrightarrow{ob} commit(i')$.
- $W(r) = w$, if the read action r reads the value written by the write action w.
- $V(w) = v$, if the write action w writes the value v.
- A release action i *synchronizes-with* an acquire action i' if $MemLoc(i) = MemLoc(i')$ and $i \xrightarrow{so} i'$.
- Happens-before relation is the transitive closure of synchronizes-with and program order relations.

It should be noted that there can be multiple complete trace executions corresponding to a single abstract execution. All such executions lead to the same final observable state.

4 Relaxed Behaviors Allowed by WOMM

Initially, `x == y == 0`

Proc 1:	Proc 2:
`r1 = x;`	`r2 = y;`
`y = 1;`	`x = 1;`

`r1 == r2 == 1` is allowed by WOMM

Fig. 8. Behavior Allowed by WOMM

A large class of hardware optimizations can be described as reordering of *locally independent actions*, simply referred as *independent actions*. In the code fragment from Figure 8, the processors can reorder the writes before the reads as they access different memory locations, resulting in the behavior described. WOMM allows this behavior in the trace execution $\langle issue(\texttt{r1} = \texttt{x}), issue(\texttt{r2} = \texttt{y}), issue(\texttt{y} = \texttt{1}), issue(\texttt{x} = \texttt{1}), commit(\texttt{y} = \texttt{1}), commit(\texttt{x} = \texttt{1}), commit(\texttt{r1} = \texttt{x}), commit(\texttt{r2} = \texttt{y})\rangle$. Note that the semantics allow this trace. Before the commit of read of x in Proc 1, the pomset of `x` contains three items — two write items $WI\langle 0, \top\rangle$, $WI\langle 1, 2\rangle$ and a read item $DI\langle 1\rangle$. The relations in the pomset are as follows: $\{WI\langle 0, \top\rangle \prec WI\langle 1, 2\rangle, WI\langle 0, \top\rangle \prec DI\langle 1\rangle\}$. From the definition of *ConsistentRead* (Figure 5), the read can see the value `1`. Similarly, the read of `y` can also see the value `1`, thus making this behavior possible. The behavior in Figure 1 is also allowed by WOMM in a similar way.

We present the formal definition of independent actions as follows.

Definition 1 (Independent Actions). *Let s_1 and s_2 be two actions from the same processor p in an execution such that $s_1 \xrightarrow{po} s_2$. They are* independent *if all of the following are true:*

1. *s_1 and s_2 are not conflicting accesses.*
2. *There is no data flow from s_1 to s_2 through a register.*
3. *s_1 is not an acquire.*
4. *s_2 is not a release.*
5. *Both s_1 and s_2 are not synchronization instructions.*
6. *s_1 is not a conditional branch.*

The following theorem states that a processor can reorder independent instructions at runtime under WOMM semantics.

Theorem 1. *Let at any point of an execution E, s be an action issued by a processor p but not yet committed. The action s can commit immediately if all the uncommitted actions by processor p are independent with s.*

Proof. Proof of this theorem follows directly from the definition of independent actions and the semantics of commit (Figure 7). □

The memory subsystems and the communication networks of relaxed multiprocessor platforms often do not guarantee strict ordering of "events". In the example of Figure 2, the writes by Proc 1 are seen in different orders by the reads of Proc 2 and Proc 3. WOMM allows this behavior by committing the writes from Proc 1 before any reads. As Proc 1 is not synchronized with Proc 2 or Proc 3, the reads can see any of the writes from Proc 1. Note that this behavior cannot be produced by simply reordering independent actions.

In Section 5 we show that WOMM is weaker than the traditional hardware memory models such as TSO, PSO, RMO etc. This implies that WOMM allows all runtime optimizations allowed by those models.

5 Relationship with Other Memory Models

In this section, we show that WOMM is strictly weaker than the Location Consistency (LC) memory model [7]. As the LC is weaker than many common hardware memory models (such as TSO, PSO [2], Release Consistency [10]), this in turn, makes WOMM weaker than those traditional memory models. Hence, any runtime optimizations allowed by the these hardware memory models are also allowed by WOMM.

Theorem 2. *WOMM is strictly weaker than the Location Consistency memory model.*

Proof. As the LC semantics do not allow explicit reordering of instructions, an LC execution [7] can be viewed as a trace execution, where an issue of an instruction is immediately followed by the commit of the same and the ordering of those instructions is consistent with the program and synchronization order. The write and the synchronization instructions update the pomset the same way as WOMM's *IssueUpdate* (Figure 3) and *CommitUpdate* (Figure 6) functions. A read instruction in LC gets its value from the pomset, restricted by the following constraints — the write item is either not ordered before the read, or there is no other intervening write item in the pomset relation. As the instructions are not reordered and the execution respects all uniprocessor control dependencies, it is not possible for a read to see a write item which is ordered after the read. Hence the constraints imposed on read by the LC are equivalent to the *ConsistentRead* function. Hence it is easy to see that this trace is allowed by the WOMM. As a result, given a program, any observable state reachable under the LC semantics is also reachable under WOMM.

The example in Figure 8 shows that WOMM is strictly weaker than the LC. In the LC, as at least one read must commit before the writes with value `1`, both

`r1` and `r2` cannot be `1` simultaneously. But this is possible in WOMM as the writes with value `1` can commit first followed by the reads and both the reads can see the value `1`. □

6 The DRF Guarantee

Designers of memory models face two conflicting goals — on one hand, the semantics should be strong enough so that it is easy for the programmers to understand it (e.g. the SC); on the other hand, it should be weak enough to allow different optimizations. A compromise between these two goals is reached via the *DRF guarantee*.

Definition 2 (Datarace-Free Program). *In an execution, a pair of instructions accessing the same shared-memory location are called* conflicting accesses *if at least one of them is a write. A (closed) program is called* datarace-free *(DRF) or* correctly synchronized *if all pairs of conflicting accesses in any sequentially consistent execution of that program are ordered by* happens-before.

Definition 3 (DRF Guarantee). *Relaxed memory models with the* DRF guarantee *ensure that any execution of a datarace-free program under the relaxed memory model is equivalent to some sequentially consistent execution of that program.*

This property is highly desirable for weak memory models as it ensures that programmers need not worry about the weak memory models as long as they write only correctly synchronized programs. In this section, we prove that WOMM gives the DRF guarantee. To show that WOMM has this property, we first state the following lemma in context of an abstract execution allowed by WOMM.

Lemma 1. *In a WOMM execution of a correctly synchronized program, if each read sees a write that happens-before the read, then the execution has sequentially consistent behavior.*

This lemma is proved in [9] for the Java Memory Model. The same proof can be used in the context of an abstract WOMM execution.

The following lemma states a connection between the relation of pomset items at any state of a WOMM trace execution and the happens-before relation of the abstract execution equivalent to the trace.

Lemma 2. *Consider a WOMM trace execution E. Let I and I' be two complete pomset items, corresponding to the instructions i and i' respectively, belonging to the same memory location at any state Ω during E. Let E_A be the abstract execution equivalent to E. $I \prec I'$ in Ω iff $i \xrightarrow{hb} i'$ in E_A.*

This lemma is proved in Appendix A.

Now we prove the following theorem stating that WOMM provides the DRF guarantee.

Theorem 3. *Any abstract execution of a correctly synchronized program allowed by WOMM is equivalent to a sequentially consistent execution.*

Proof. Let E_A be an abstract execution of a correctly synchronized program P allowed by WOMM and E be a trace execution of P equivalent to E_A. By Lemma 1, if E_A is not equivalent to any SC execution of P, there must be at least one read that sees a write that does not happen-before it. Using Lemma 2, when such a read commits in E, the corresponding read item I_D will not be related to the write item I_W it sees. Let r be the first such read, which sees a unrelated write w, to commit in E. Using Lemma 2, all reads committed before r see writes that happen-before them.

We construct a complete trace execution E' from E the following way. Let A' be the set of last committed instructions from each processor (in *ob* order) when r is committed in E. We then include in A' all instructions of E that precede those instructions in program order. Some of such instructions may not be committed when r is committed in E. In E', those instructions are committed in program order such that each read sees a write that happens-before it (that is, the corresponding read item is ordered after the seen write item in the pomset). Note that none of these incomplete instructions can be an acquire instruction, according to the semantics of commit. Hence, instructions from different processors can be committed in any order consistent with program order, without affecting the happens-before relations. At last, the read r is modified such that it also sees a write that happens-before it. Note that, this completion does not affect the already committed instructions, neither does it introduce any new *sw* edge.

Let us consider the execution E'_A corresponding to the complete execution E'. In E'_A, each read sees a write that happens-before it. Hence, by Lemma 1, it must be equivalent to an SC execution, but there is a write w and a read r to the same memory location which are not related by happens-before. This contradicts the fact that the program is correctly synchronized. Thus, there cannot exist any read in E_A which sees a write that does not happen-before the read. Hence, by Lemma 1, the execution E_A is equivalent to an SC execution. □

7 Related Work

Most of the early works in relaxed memory models for hardware multiprocessors has been summarized in [3]. Shen et al [11] shows that breaking an instruction into finer-grained operations and using an abstract semantic cache can lead to weak memory models. The notion of abstract shared memory state used in the Location Consistency memory model [7] is very similar to our memory model. A recent work [6] developed rigorous axiomatic semantics of multiprocessor machine code for x86 architectures.

Some recent works have stressed the need for precise, concrete, operational style specification of memory models [12,13]. Boudol et al [14] shows how operational specification of memory models help in proving the DRF guarantee for such models.

Verifying concurrent programs under relaxed memory models requires precise specification of the memory model. Burckhardt et al [15] exhaustively check

all executions of a concurrent program under a relaxed memory model that allows certain reordering and buffering. In our earlier works, we formalized the memory model for C# [16] and proposed an operational approximation of the Java Memory Model [17] and developed memory model sensitive model checkers to find bugs in concurrent C# and Java programs.

8 Conclusion and Future Work

In this paper, we have presented a weak operational multiprocessor memory model. Our work is useful for understanding such models at an abstract level. The compiler writers and low-level programmers can follow this semantics without going into the details for each hardware memory model separately. We have shown that our proposed semantics is weak enough to allow many hardware optimizations. We formally compared it with the Location Consistency model. The operational style of our specification enables traversal and enumeration of executions of a given program that are allowed by the model, helping in easy integration into model checkers.

In future, we plan to extend our memory model to also allow speculative execution of instructions. We also plan to integrate the current model to state space exploration tools (such as software model checkers) for memory model sensitive program reasoning.

References

1. Lamport, L.: How to make a multiprocessor computer that correctly executes multiprocess programs. IEEE Trans. Comput. 28(9), 690–691 (1979)
2. Weaver, D.L., Germond, T.: The SPARC Architecture Manual: Version 9. Prentice Hall, Englewood Cliffs (1994)
3. Adve, S.V., Gharachorloo, K.: Shared memory consistency models: A tutorial. IEEE Computer 29(12), 66–76 (1996)
4. Intel: Intel® 64 and ia-32 architectures software developer's manual volume 3a: System Programming Guide, http://www.intel.com/Assets/PDF/manual/253668.pdf
5. AMD: Amd64 architecture programmer's manual (2007)
6. Sarkar, S., Sewell, P., Nardelli, F.Z., Owens, S., Ridge, T., Braibant, T., Myreen, M.O., Alglave, J.: The semantics of x86-cc multiprocessor machine code. In: POPL 2009: Proceedings of the 36th annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pp. 379–391. ACM, New York (2009)
7. Gao, G.R., Sarkar, V.: Location consistency — a new memory model and cache consistency protocol. IEEE Trans. Comput. 49(8), 798–813 (2000)
8. Intel: A formal specification of intel ® itanium ® processor family memory ordering (October 2002), http://www.intel.com/design/itanium/downloads/251429.htm
9. Manson, J., Pugh, W., Adve, S.V.: The java memory model. In: POPL 2005: Proceedings of the 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pp. 378–391. ACM, New York (2005)

10. Gharachorloo, K., Lenoski, D., Laudon, J., Gibbons, P., Gupta, A., Hennessy, J.: Memory consistency and event ordering in scalable shared-memory multiprocessors. In: Proceedings of the Annual International Symposium on Computer Architecture, pp. 15–26 (May 1990)
11. Shen, X., Arvind, Rudolph, L.: Commit-reconcile & fences (crf): a new memory model for architects and compiler writers. SIGARCH Comput. Archit. News 27(2), 150–161 (1999)
12. Nardelli, F.Z., Sewell, P., Sevcik, J., Sarkar, S., Owens, S., Maranget, L., Batty, M., Alglave, J.: Relaxed memory models must be rigorous. In: Exploiting Concurrency Efficiently and Correctly Workshop (2009)
13. Mador-Haim, S., Alur, R., Martin, M.M.: Specifying relaxed memory models for state exploration tools. In: Exploiting Concurrency Efficiently and Correctly Workshop (2009)
14. Boudol, G., Petri, G.: Relaxed memory models: an operational approach. In: POPL 2009: Proceedings of the 36th Annual ACM SIGPLAN-SIGACT Smposium on Principles of Programming Languages, pp. 392–403. ACM, New York (2009)
15. Burckhardt, S., Alur, R., Martin, M.M.K.: Checkfence: checking consistency of concurrent data types on relaxed memory models. In: PLDI 2007: Proceedings of the 2007 ACM SIGPLAN Conference on Programming Language Design and Implementation, pp. 12–21. ACM, New York (2007)
16. Huynh, T.Q., Roychoudhury, A.: Memory model sensitive bytecode verification. Form. Methods Syst. Des. 31(3), 281–305 (2007)
17. De, A., Roychoudhury, A., D'Souza, D.: Java memory model aware software validation. In: PASTE 2008: Proceedings of the 8th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering, pp. 8–14. ACM, New York (2008)

A Proof of Lemma 2

Proof. If $I \prec I'$ at some state Ω in E, then, from the definition of *IssueUpdate* (Figure 3) and *CommitUpdate* (Figure 6), either i and i' are from the same processor and $issue(i) \xrightarrow{ob} issue(i')$, or there is a sequence of synchronization instructions $s_1, s'_1, \ldots, s_n, s'_n$ such that each s_j and s'_j are release and acquire instructions on the same memory location respectively, s'_j and s_{j+1} are from the same processor, $issue(i) \xrightarrow{ob} issue(s_1) \xrightarrow{ob} commit(s_1) \xrightarrow{ob} commit(s'_1) \ldots commit(s'_n) \xrightarrow{ob} commit(i')$ and $issue(s'_n) \xrightarrow{ob} issue(i')$ (i might be same as s_1 and s'_n might be same as i'). From Section 3.6, it implies that in E_A, $i \xrightarrow{po} s_1 \xrightarrow{sw} s'_1 \ldots s'_n \xrightarrow{po} i'$. From the definition of happens-before relation, $i \xrightarrow{hb} i'$.

Conversely, if $i \xrightarrow{hb} i'$, either $i \xrightarrow{po} i'$, or there are instructions $i_1, i'_1, \ldots, i_n, i'_n$ such that $i \xrightarrow{po} i_1 \xrightarrow{sw} i'_1 \xrightarrow{po} i_2 \ldots i_n \xrightarrow{sw} i'_n \xrightarrow{po} i'$, where each i_j is a release and i'_j is an acquire action. In the first case, clearly $I \prec I'$. In the second case, from the semantics of commit, when i' commits, all these instruction must already be committed. From *IssueUpdate* and *CommitUpdate*, the pomset items of each of these instructions are ordered after the pomset item of the previous instruction. Hence, $I \prec I'$. □

A Memory Model for Static Analysis of C Programs

Zhongxing Xu[1], Ted Kremenek[2], and Jian Zhang[1]

[1] State Key Laboratory of Computer Science
Institute of Software
Chinese Academy of Sciences
xzx@ios.ac.cn,
zj@ios.ac.cn
[2] Apple Inc.
kremenek@apple.com

Abstract. Automatic bug finding with static analysis requires precise tracking of different memory object values. This paper describes a memory modeling method for static analysis of C programs. It is particularly suitable for precise path-sensitive analyses, e.g., symbolic execution. It can handle almost all kinds of C expressions, including arbitrary levels of pointer dereferences, pointer arithmetic, composite array and struct data types, arbitrary type casts, dynamic memory allocation, etc. It maps aliased lvalue expressions to the identical object without extra alias analysis. The model has been implemented in the Clang static analyzer and enhanced the analyzer a lot by enabling it to have precise value tracking ability.

1 Introduction

Recently there has been a large number of works on bug finding with symbolic execution technique. In these works, tracking values of different memory objects along a single path is a common requirement. Some works get the run-time addresses of memory objects by actually compiling and running the program [1] [3]. These are dynamic techniques. Programs being checked must be instrumented and linked with an auxiliary library and run. Other works solve the memory object identifying problem through various static ways. The simplest approach is to only track simple variables with names, and ignore multi-level pointers, array elements, and struct fields. This would surely sacrifice much analysis power.

This paper proposes a memory modeling method that is particularly suitable for symbolic execution of C programs. It enables the symbolic execution to identify and track each memory object precisely. We give algorithms that enable the mapping from C l-value expressions to memory objects during the analysis. Thus no separate alias analysis is required.

Memory model is the way that the analysis tool models the storage of the underlying machine on which the code runs. It is the basis of language semantics simulation and a key component of static code analysis tools.

T. Margaria and B. Steffen (Eds.): ISoLA 2010, Part I, LNCS 6415, pp. 535–548, 2010.

Memory model determines how accurately the tool can simulate the language semantics and thus affects the ability of the tool to detect bugs. Surprisingly, few papers addressed the memory modeling issue in the static analysis field.

Unlike other high level programming languages, the C programming language assumes a rather low level memory model and provides extensive kinds of memory operations, such as multi-level pointers, arbitrary pointer arithmetic, built-in array and struct data types. Pointers in C can point to arbitrary locations of the memory, and can be cast freely between different types. All these make it difficult to simulate the C semantics accurately.

In Section 2 and 3, we introduce two basic memory models which are commonly used but have some limitations. In Section 4 we describe our novel memory model, which can precisely map each l-value expressions to the corresponding memory object. In Section 5 we describe how to simulate the C language semantics with the new memory model. We give some examples and implementation in Section 6 and 7. We compare with related works in Section 8 and conclude in Section 9.

2 Name Binding Model

The name-binding model is the most basic storage model present in the semantics textbooks. In this model, the computer memory is seen as name-value pairs. When an assignment expression is evaluated, we bind the name of the variable on the left-hand-side to the value of the expression on the right-hand-side. While this model is very common, it is not powerful enough to be used for simulating the C semantics. Consider the following example:

```
int x, y;
int *p = &x;
x = 3;
*p = 4;
y = x;
```

The C language has pointers. In this example, `p` is a pointer variable that points to the variable `x`. The presence of pointers brings the aliasing problem. The aliasing problem is that two or more names can represent the same storage location. In this example, `*p` and `x` are aliases.

The name-binding model cannot deal with the aliasing problem. When we modify the value of a name, we should also modify the value of all names aliased to it accordingly. But in no way can we know the aliases of a name in the name-binding model.

3 Array Simulation Model

One of the deficiencies of the name-binding model is that it lacks the concept of memory locations. The pointers in C, however, are invented to manipulate memory locations.

Zhang proposed an array model for the memory [11]. The motivation of the array model is simple: intuitively the memory can be seen as a large array. If we allocate all variables on an array, all operations on variables can be transformed into operations on corresponding array elements, and most importantly, the array element indices can be taken as the memory locations for the variables.

The example in Section 2 can be transformed using the array memory model:

```
Assume mem[] is the memory simulation array.
Memory allocation:
  x: mem[1], y: mem[2], p: mem[3]

mem[3] = 1; // p = &x; mem[3] is 'p',
            // 1 is x's simulation location.
mem[1] = 3; // x = 3;
mem[mem[3]] = 4; // dereference '*' means
                 // we have to nest mem[]'s
mem[2] = mem[1]; // y = x;
```

The array model is somewhat more powerful than the name-binding model. It can solve some problems in program analysis [10]. But its disadvantages are also obvious.

The array model requires that every variable has a fixed position. This can be achieved as long as we know the exact size of every memory object. Once we have a memory object of unknown size, such as a variable-length array or a heap object of unknown size, the array model is unusable.

A slight improvement for the array model is that instead of using a single large array, we use multiple arrays. Each memory object has a corresponding array for it. But composite memory objects are difficult to represent in this model. For example, for object struct array `struct s { int d } sa[2];`, if we use a single array to represent it, it still has the same weakness as the single array model. If we use multiple arrays to represent it, we will lose the hierarchy relation among memory objects.

4 Region Based Ternary Model

Formal semantics models program state with two mappings [8]: a variable environment that associates a location with each variable and a store that associates a value with each location. Formally, we define a variable environment Env as a mapping of

$$Env = Var \rightarrow Loc$$

where Loc is a set of locations. A store is the mapping of

$$Store = Loc \rightarrow Value$$

The name-binding model in Section 2 lacks the concept of locations. The array model in Section 3 does have the concept of locations. It uses concrete integers

to represent locations. This concretization of location limits the applicability of the array model.

In this section we develop a new representation of locations: regions. A *region* is an abstract chunk of memory corresponding to an lvalue expression in the C programming language.

According to the C standard [5], an *lvalue* is an expression with an object type. That is, an lvalue expression has an associated memory object. We use an abstract region to represent this memory object.

Thus in our region based memory model, every lvalue expression should have a corresponding region. Furthermore, lvalue expressions indicating the same memory object should have the same region corresponding to them. Next we describe the way to get the region associated with an lvalue expression.

4.1 Region Hierarchy

We define several kinds of regions. For explicitly declared variables, we have *VarRegion* identified by the variable declarations. Every variable has a unique VarRegion associated with it.

If the variable is of array or struct type, it has subobjects called element or field. To represent this hierarchy between memory objects, we introduce the concept of subregions. A region can be the subregion of another region. There is a super region pointer pointing to the super region of a subregion.

For array elements, we have *ElementRegion* with its super region pointer pointing to its array region. Likewise, for struct fields, we have *FieldRegion* with its super region pointer pointing to its struct region. ElementRegions are identified by their array regions and the indices. FieldRegions are identified by their struct regions and the field declaration.

In C, there are three kinds of storage classes: local (stack), global (static), dynamically allocated (heap). We also have a *MemSpaceRegion* for this concept. There are three MemSpaceRegions for stack, heap, and static storage respectively. All local variables have the stack MemSpaceRegion as their super region. All global variables have the static MemSpaceRegion as their super region. All dynamically allocated objects (mostly by malloc()) have the heap MemSpaceRegion as their super region.

In static analysis, we sometimes would have symbolic values. For example, we assume function arguments and global variables holds symbolic values at the entry of the function. If the symbolic variable is a pointer, it may point to some unknown memory block. For this case, we have *SymbolicRegion* for representing the memory block pointed to by the symbolic pointer. SymbolicRegions are identified by the symbolic pointer values that point to them.

4.2 Region Properties

Besides storage classes, memory objects have extents, or sizes. Some objects' extents are explicit. For example, a char variable has the extent of one byte. But dynamically allocated objects can have various extents. We record this

information with a region-extent mapping from the object's associated region to its extent. The extents can be in various forms: concrete integer value or symbolic unknown value.

Memory state is modeled by the bindings of the regions. There are two kinds of bindings: direct binding and default binding. After an assignment expression, like `x = 3;`, we set the direct binding of region of `x` to `3`. Default binding is usually set on super regions. If a subregion has no binding, then it could use its super region's default binding as its value. For example, after function call `bzero(buf)`, we can set the default binding of the region pointed to by `buf` to 0. Without default binding, we have to set each element of that region to 0, which is prohibitively expensive for large arrays.

4.3 Region Views

The C programming language permits arbitrary conversions between types of pointers. This poses great difficulty to static analysis tools.

Consider the following code snippet:

```
void *p = malloc(10);
char *buf1 = (char *) p;
buf1[0] = 'a';

int *buf2 = (int *) p;
buf2[0] = 0;

char c = buf1[0];
```

This is a contrived example. But the code pattern is fairly common in system programs. Programmers often allocate a generic block of memory, then cast it to different types for different uses. How do we deal with such (ab)uses of the C type system?

The essence of this problem is that in C we can have typeless generic chunk of memory. In the code example above the dynamically allocated memory pointed to by `p` is such a chunk of memory. The later casts from it to `char*` and `int*` can be interpreted as installing "views" to it. When it is operated by pointer `buf1`, it is viewed as a memory block of type `char`. When it is operated by pointer `buf2`, it is viewed as a memory block of type `int`.

We set up a region view mapping from a region to its various views: some anonymous typed region. Casting a generic block of memory to some type is equivalent to adding a new view to the block of memory. We create an anonymous typed region to represent the view. Later when the generic memory block is cast to another type, a new anonymous typed region is created to represent this new view. The idea is illustrated in Figure 1.

A region can have multiple views simultaneously. But only one view can be in effect at a time. When the region is operated on by one view region of it, the information associated with its other view regions must be invalidated.

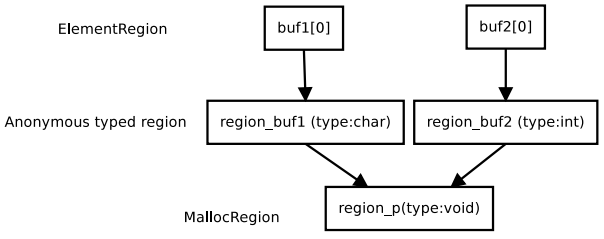


Fig. 1. Anonymous typed region illustration. Region region_p is the generic memory block pointed to by p. Region region_buf1 is the same memory block with type char. Region region_buf2 is the same memory block with type int. Regions buf1[0] and buf2[0] are the ElementRegions on regions region_buf1 and region_buf2.

Returning to the example, after expression `buf1[0] = 'a';`, `buf1[0]` has value `'a'`. But after expression `buf2[0] = 0`, we not only set the value of `buf2[0]` to `0`, but also removes the binding for `buf1[0]`. When `c` is assigned `buf1[0]`, we can spot that `buf1[0]` is an undefined value.

5 Simulation of C Semantics

With the region based memory model, we can simulate the C semantics precisely. We still use the semantics model in Section 4. But we extend it as follows.

The program state is modeled by two mappings: Environment and Store. We define Environment as a mapping from expressions to values:

$$Env : Expr \rightarrow SVal$$

where Expr is an element of C expressions. SVal is some kind of abstract values, which we will describe in Section 5.1. The Store is defined as a mapping from Regions to values:

$$Store : Region \rightarrow SVal$$

where Region is the abstract representation of memory objects.

5.1 Abstract Values

We divide the values occurring in the symbolic simulation into two classes: locations *loc* and non-locations *non-loc*. A *loc* can be a region or unknown. *loc* represents an abstract memory location. Pointers in C all have *loc* values. The rests are non-location values *non-loc*. A *non-loc* can be a concrete integer, a symbolic integer, etc.

5.2 l-Value and r-Value

The C programming language standard [5] classifies expressions into lvalue and rvalue. Expressions referring to objects are lvalue expressions. The rests are

rvalue expressions. To evaluate expressions, we have to distinguish between an expression's l-value and r-value. We define an expression's l-value to be the memory location of its associated memory object. An expression's r-value is the value associated with the memory object if the expression is an lvalue expression or the semantic value if the expression is not an lvalue expression.

Note that lvalue expressions have both l-value and r-value. Non-lvalue expressions only have r-value, i.e., their semantic value. l-value can only be *loc*s, while r-value can be both *loc*s and *non-loc*s. For example, `*p`'s r-value is a *loc*.

5.3 Evaluation Rules

In this section we give some rules of evaluation of C expressions. We define some notations: l-value(e) returns expression e's l-value, which is a loc. r-value(e) returns expression e's r-value, which is a *loc* or *non-loc*. VarRegion(v) return the unique region associated with variable declaration v. ElementRegion(r, i) returns the unique region associated with the i'th element of array region r. Similarly, FieldRegion(r, f) returns the unique region associated with the f field of struct r. $Store(loc)$ returns the value associated with location *loc*.

Evaluation rules for main kinds of expressions of C are specified in Table 1.

Table 1. Evaluation rules for C expressions

	l-value	r-value
constant n	N/A	n
variable x	VarRegion(x)	Store(l-value(x))
array a	VarRegion(a)	N/A
`a[e]`	ElementRegion(l-value(a), r-value(e))	Store(l-value(a[e]))
`s.d`	FieldRegion(l-value(s), d)	Store(l-value(s.d))
`p->d`	FieldRegion(r-value(p), d)	Store(l-value(`p->d`))
`&expr`	N/A	l-value(expr)
`*expr`	r-value(expr)	Store(r-value(expr))
`alloca(n)`	N/A	AllocaRegion(n)
`malloc(n)`	N/A	MallocRegion(n)

The region-base memory model supports almost all kinds of C expression semantics: arbitrary level pointer, composite struct and array data types, dynamic memory allocation, and symbolic pointer.

For pointer arithmetic, we assume that it only happens to element regions. We get the index of the element region, apply the offset to it, and get an element region with the new index.

For dynamically allocated memory, we create a MallocRegion with the size and an execution counter to differentiate memory chunks allocated in the same statement.

Let us look at an example showing how the evaluation rules are applied. Consider code snippet:

```
struct s1 {
  int d;
};

struct s2 {
  struct s1 *p;
};

void foo(void) {
  struct s1 data;
  struct s2 *sp;
  int a[2];

  sp = malloc(sizeof(struct s2));
  sp->p = &data;
  sp->p->d = 3;
  a[1] = data.d;
}
```

After we have processed the three variable declarations in function `foo`, we have program state shown as follows:

Expression	Region	Value
data	region 1	N/A
data.d	region 2	undefined
sp	region 3	undefined
a	region 4	N/A
a[0]	region 5	undefined
a[1]	region 6	undefined

When processing the malloc statement, we will have a new region in the program state. The updated program state is shown as follows, where $MallocReg$ represents the memory region allocated by `malloc()`.

Expression	Region	Value
data	region 1	N/A
data.d	region 2	undefined
sp	region 3	**region 7**
a	region 4	N/A
a[0]	region 5	undefined
a[1]	region 6	undefined
$MallocReg_0$	**region 7**	**N/A**
$MallocReg_0.p$	**region 8**	**undefined**

Next we process statement `sp->p = &data;`. Evaluating expression `&data` requires the l-value of `data`, which is region 1. Then we get the r-value of `sp`, which is region 7. getFieldRegion(region7,p) returns region 8. The updated program state is shown as follows:

Expression	Region	Value
data	region 1	N/A
data.d	region 2	undefined
sp	region 3	region 7
a	region 4	N/A
a[0]	region 5	undefined
a[1]	region 6	undefined
$MallocReg_0$	region 7	N/A
$MallocReg_0.p$	region 8	**region 1**

Next we process statement `sp->p->d = 3;` Similarly we get the r-value of `sp->p`, which is region1, and the l-value of `sp->p->d`, which is region2. The updated program state is shown as follows:

Expression	Region	Value
data	region 1	N/A
data.d	region 2	**3**
sp	region 3	region 7
a	region 4	N/A
a[0]	region 5	undefined
a[1]	region 6	undefined
$MallocReg_0$	region 7	N/A
$MallocReg_0.p$	region 8	region 1

Then we process statement `a[1] = data.d;` The l-value of `data.d` is region2, then we get its r-value 3. The l-value of `a[1]` is region 6. The updated program state is shown as follows:

Expression	Regions	Values
data	region 1	N/A
data.d	region 2	3
sp	region 3	region 7
a	region 4	N/A
a[0]	region 5	undefined
a[1]	region 6	**3**
$MallocReg_0$	region 7	N/A
$MallocReg_0.p$	region 8	region 1

From the above example we can see that all memory objects are represented unambiguously, and their corresponding regions are computed on the fly with little overhead. The region hierarchy is shown in Figure 2.

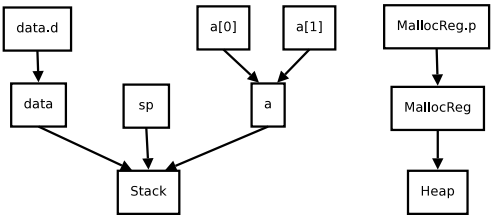


Fig. 2. The final memory region hierarchy. The arrow points to the super region.

6 An Example

In this section we use an example to show the power of the region-base memory model. Consider the following code snippet.

```
struct s { int data[2]; }
void f(struct s buf) {
1  int i = 3, *q = NULL;
2  struct s* p;

3  if (buf.data[1] == 1)
4    q = &i;

5  p = &buf;

6  if (p->data[1] == 1)
7    p->data[0] = *q;

8  return;
}
```

In this code, if we do not track the value of the struct field `buf.data[1]`, we would have a path leading to the NULL pointer dereference of `q` at line 7: 1,2,3,5,6,7,8.

With the region memory model, we can precisely know that `buf.data[1]` and `p->data[1]` refer to the identical memory object. Thus the path condition along the previous path cannot be satisfied: `buf.data[1]` cannot be simultaneously equal and unequal to 1.

The region hierarchy and storage mapping is shown in Figure 3. When `buf.data[1]` is evaluated, we get the region associated with `buf`, `buf.data`, `buf.data[1]` respectively. Then the value of `buf.data[1]` is retrieved, which is a symbolic value $1. When `p->data[1]` is evaluated, we first get `p`'s rvalue, which is the memory region of `buf`. Then along the same way, we get the value for `buf.data[1]`, which is the symbolic value $1.

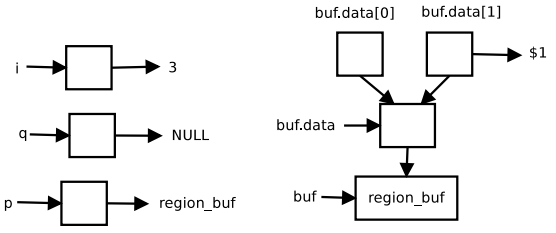


Fig. 3. The region hierarchy and storage mapping. Squares represent memory regions. $1 is the symbolic value of buf.data[1]

The for path 1,2,3,5,6,7,8, we get path condition:

$$\$1! = 1 \wedge \$1 == 1,$$

which cannot be satisfied. There will be no false alarm for the NULL pointer dereference of `q` at line 7.

7 Implementation

The region based memory model is fully implemented in the Clang analyzer [2]. Clang is a new C, C++ and Objective C front-end for the LLVM [7] compiler. The static analyzer is an official part of Clang that find bugs in C and Objective-C programs. The second author of this paper is the original architect of the static analyzer core [6]. The analyzer has very good modularity. It is designed in such a way that main components can be swapped in and out. Basically it has the following components:

- The core engine, driving the analysis through the program in some order.
- State manager, managing the simulated program states.
- Constraint manager, recording and solving path conditions collected along a program path.
- Store manager, modeling the program storage.

The region based memory model is implemented as the region store manager in the tool. It adds the full field sensitivity to the analysis.

An implementation feature that is worth mentioning is the lazy binding technique we employed. It can be best illustrated by an example. Consider code snippet:

```
for (...) {
  ...
  if (...) {
L:    int buf[8096];
  }
}
```

As we are using the region based memory model, theoretically we have a separate region for each of the array elements on line `L`. Since `buf` is a local array, we have to initialize all of its elements to have value *undefined*. Because we could have multiple paths leading to the definition of `buf`, we have to initialize its 8096 elements multiple times. This proved very time consuming. Moreover, few of the elements are actually used during later analysis.

To solve this problem, we switched to an implementation that employed lazy initialization. We do not initialize any variables. Instead, we compute their initial value the first time it is used according to their storage classes. This optimization reduced the analysis cost dramatically.

8 Related Work

Hampapuram et al. used a region-based memory model in their work [4]. Our model is different than theirs in various ways. The main feature of our model is the region hierarchy, which plays an important role in tracking and distinguishing memory objects and reasoning about their relations.

Saturn [9] is a general framework for building precise and scalable static error detection systems. It models program operations with Boolean satisfiability (SAT) techniques down to the bit level. Pointers in Saturn are modeled with Guarded Location Sets (GLS). The GLS method essentially gives each location an explicit name and records the set of locations a pointer could reference at a particular program point. This memory representation is not as rich as our region-based memory model, which treats memory objects as first-class values with properties such as extents and relationships to other memory regions (e.g., region views to capture type-casting relationships). One consequence of these differences is that Saturn's strictly name-based memory model does not amend itself well to reasoning about pointer arithmetic nor array operations, both of which are handled naturally and precisely in our region-based model (i.e., by reasoning about the indices of ElementRegions).

Other bug finding tools like EXE [1] and DART [3] circumvent the memory modeling problem by actually compiling and running the program being checked. They use the run-time addresses of memory objects to distinguish them.

The closest related work to ours in the traditional program analysis field is alias analysis. We both try to solve the similar problem: the correspondence between expressions and memory objects. The requirements, however, are different.

Alias analysis for compiler optimization aims for a conservative result. It usually computes "may" aliases. We aim to get more precise alias relation. Due to the path sensitive character of our application, we do have more precise program state information that enables us to get more precise alias information.

On the usage aspect, alias analysis is often used as a separate analysis pass on the program. Then its results are stored for later use. Our memory model is used in combination with the whole symbolic analysis of the program. The corresponding memory regions are computed on the fly during the symbolic analysis. There is no separate "region analysis" pass for our analysis model.

9 Conclusion

We designed a region-based memory model for path-sensitive symbolic program analysis. In summary, the memory model has the following features which make it more powerful and suitable for symbolic execution:

- It can handle almost all kinds C expressions, including arbitrary levels of pointer dereferences, pointer arithmetic, composite array and struct data types, arbitrary type casts, dynamic memory allocation, etc.
- It maps aliased lvalue expressions to the identical object without extra alias analysis.
- It can represent various properties of memory objects: known and unknown extent, storage class, hierarchy relation, concrete and symbolic values, etc.

The memory model has been implemented in the Clang analyzer [2]. Our preliminary experimentation showed that it is effective for adding field sensitivity to our static analysis tool.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China (Grant No. 60633010, 60903049) and the High-Tech (863) program of China (Grant No. 2009AA01Z148).

References

1. Cadar, C., Twohey, P., Ganesh, V., Engler, D.: EXE: A system for automatically generating inputs of death using symbolic execution. Technical report, Computer System Laboratory, Stanford University (2006)
2. Clang static analyzer, http://clang-analyzer.llvm.org/
3. Godefroid, P., Klarlund, N., Sen, K.: DART: Directed automated random testing. In: Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation, pp. 213–223 (2005)
4. Hampapuram, H., Yang, Y., Das, M.: Symbolic path simulation in path-sensitive dataflow analysis. In: Proceedings of the 6th ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering, pp. 52–58 (2005)
5. WG14 ISO/IEC 9899:201x, editor. Programming Languages - C. ISO (1999), http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1336.pdf
6. Kremenek, T.: Finding bugs with the clang static analyzer, http://llvm.org/devmtg/2008-08/Kremenek_StaticAnalyzer.pdf
7. The LLVM compiler infrastructure, http://llvm.org/
8. Nielson, H.R., Nielson, F.: Semantics with applications: a formal introduction. John Wiley & Sons Inc., Chichester (1992)

9. Xie, Y., Aiken, A.: Saturn: A scalable framework for error detection using Boolean satisfiability. ACM Transactions on Programming Languages and Systems 29(3) (2007)
10. Xu, Z., Zhang, J.: A test data generation tool for unit testing of C programs. In: Proceedings of the International Conference on Quality Software, pp. 107–116 (2006)
11. Zhang, J.: Symbolic execution of program paths involving pointers and structure variables. In: Proceedings of the Fourth International Conference on Quality Software, pp. 87–92 (2004)

Analysing Message Sequence Graph Specifications

Joy Chakraborty[1], Deepak D'Souza[2], and K. Narayan Kumar[3]

[1] Motorola India Private Limited
C.V. Raman Nagar
Bangalore 560093, India
j.chakraborty@motorola.com
[2] Computer Science and Automation
Indian Institute of Science
Bangalore 560012
India
deepakd@csa.iisc.ernet.in
[3] Chennai Mathematical Institute
H1 SIPCOT IT Park
Siruseri 603103, India
kumar@cmi.ac.in

Abstract. We give a detailed construction of a finite-state transition system for a com-connected Message Sequence Graph. Though this result is well-known in the literature and forms the basis for the solution to several analysis and verification problems concerning MSG specifications, the constructions given in the literature are either not amenable to implementation, or imprecise, or simply incorrect. In contrast we give a detailed construction along with a proof of its correctness. Our transition system is amenable to implementation, and can also be used for a bounded analysis of general (not necessarily com-connected) MSG specifications.

1 Introduction

Message Sequence Chart (MSC) based specifications are a popular model of early system design, whose use is particularly widespread in the telecom and software industry. A message sequence chart describes a finite sequence, or more accurately a partially ordered sequence, of message exchanges between agents in the system. These are typically "scenarios" that a system user and developer alike can use to communicate and validate system requirements. Messages may be exchanged "synchronously" as in a handshake protocol, or "asynchronously" with separate send and receive events and a message channel to buffer undelivered messages. Message Sequence Graphs (MSG's), also sometimes referred to as "high-level" MSC's, are an activity diagram-like notation that is often used to describe infinite collections of system behaviour. They are finite graphs whose vertices are labeled by MSC's, each of which represents a single logical unit of

T. Margaria and B. Steffen (Eds.): ISoLA 2010, Part I, LNCS 6415, pp. 549–563, 2010.

interaction. The behaviours specified by an MSG are obtained by taking a path in the MSG beginning at the initial node, and collecting the behaviours given by the "concatenation" of the MSC's associated with the nodes along the path.

Given that MSC-based specifications provide an early encapsulation of system design, from an analysis and verification point of view there are some natural problems that one would like to address. Several of these have been considered in the literature, including detecting race conditions (differences in the "visual" ordering and "execution" ordering), timing conflicts, and confluence or "completability." We would like to focus on the following two problems:

1. The model-checking problem [2]: Here we are given a system description in terms of an MSG, and a property in the form of a finite-state automaton describing say undesirable behaviours. We would like to check that the system does not exhibit any of the undesirable behaviours.
2. Detecting implied scenarios [13,1]: Given a description of system behaviour in terms of an MSG, there is a natural, distributed, system model induced by the MSG. This system model is "minimal" in that any distributed implementation of the system that exhibits all the behaviours specified by the given MSG, must necessarily exhibit all the behaviours in this model. However, the minimal system model may exhibit behaviours that are outside the ones specified by the MSG: these behaviours are called *implied scenarios*. We are interested in identifying such behaviours so that the system designer can be alerted (for example to the fact that the exact behaviour specified by the MSG is not realizable by a distributed implementation).

Message Sequence Chart based specifications have received a fair amount of attention from the Computer Science theory community (see [5,6] for surveys). In particular the analysis problems mentioned above have been addressed in the following works. Alur and Yannakakis [2] show that the model-checking problem for asynchronous MSG's is undecidable in general. They propose a condition on the MSG, called "com-connectedness" (essentially that all processes that take part in any loop of the MSG must communicate directly or indirectly with each other in the loop), which is sufficient to ensure that the model-checking problem is decidable. The main task is to show that in such a case the language of behaviours defined by the MSG is regular, i.e. acceptable by a finite-state transition system. However the details of the construction are not spelt out precisely, and there is no proof of correctness given. Independently in [8], Muscholl and Peled also give a construction of a finite-state automaton for a com-connected MSG (called "loop-connected" there), in order to give decision procedures for the problems of confluence and race conditions they consider. While their construction is fairly detailed, there is no accompanying proof of correctness. Furthermore both constructions in [2,8] are not amenable to an "on-the-fly" analysis as they generate behaviours that are not part of the MSG's behaviour (these behaviours would not reach a final state in the constructed automaton). Muscholl and Peled also point to the formal language theoretic result of Clerbout and Latteux [4] which shows that the "trace-closure" of the Kleene of a regular expression R is regular provided that the dependency graph of words in R is strongly-connected, from

which the result follows. Once again this construction is not conducive to implemenation as it is done in terms of regular grammars and is aimed at a general class of semi-commutative grammar systems.

The software engineering community have parallely developed several tools and methodologies for analysing MSC-based specifications. However some of these works are based on an incorrect understanding of the result concerning regularity of com-connected MSG's. We point out some of these cases, without detracting from the several other contributions made in these papers.

- In [13], Uchitel, Kramer, and Magee claim to solve the problem of detecting implied scenarios for general (not necessarily com-connected) MSG's with synchronous messaging. This is done without explicitly building a transition system for the given MSG. No complete proofs are given in the paper or the cited technical report. This claim is incorrect as the problem is in fact undecidable (i.e for general synchronous MSG's) [3].
- In [7] Muccini gives a technique for detecting implied scenarios based on identifying "augmented" behaviours in the components of the system model for a given general synchronous MSG. The technique is validated on a few examples, but the paper gives no proofs and admits that the "correctness and completeness are still under analysis."
- The thesis of Uchitel [11] gives the construction of a finite-state transition system, called there the "trace model," for a given com-connected synchronous MSG. This construction is implemented in a tool called LTSA-MSC [10], and used as a basis for analysis in [14]. However, as we show in Sec. 6, the trace-model constructed is incomplete: it does *not* accept all behaviours specified by the MSG. As a result the tool also incorrectly flags certain behaviours of the induced system model as implied scenarios.

Our aim in this paper is to give a precise and complete description of a finite-state transition system accepting exactly the behaviours specified by a given com-connected MSG G. We give a precise description of a "reduced" transition system called $\mathcal{T}'_G$ in Sec. 4 which is guaranteed to be finite-state when the given MSG is com-connected. We give a detailed proof of correctness of our construction. Once we have such a transition system, the analysis problems we mentioned earlier can be solved easily for com-connected MSG's with synchronous messages.

It is also worth pointing out that the transition system $\mathcal{T}'_G$ we describe is sound and complete for *general* MSG's (i.e. not necessarily com-connected) as well – though of course, it may not be finite-state in this case. Our construction also handles both synchronous and asynchronous messaging in the MSG's. Further, it has no ϵ-transitions (i.e. hidden or silent transitions), and has a bounded number of transitions applicable in any state. Thus, this transition system can be used to perform a bounded analysis or model-checking to check properties like "there are no property violations by behaviours of length ≤ 15" in the given MSG model.

We thus hope that the construction we give will be a basis on which the software engineering community can build more accurate tools for analysing MSC-based specifications. Due to lack of space, we give only sketches of some of the proofs. The complete proofs are available in the technical report [3].

2 Message Sequence Charts

We begin with some preliminary notions. For a finite alphabet A, we denote the set of finite words over A by A^*. The empty word is denoted ϵ. For words u and v over A, we denote the concatenation of u followed by v by $u \cdot v$ or simply uv. We write $u \preceq v$ to denote the fact that u is a prefix of v.

A *transition system* is a tuple $\mathcal{T} = (Q, A, q_0, \rightarrow)$, Q is a set of states, A is the set of labels or alphabet of the transition system, q_0 is the initial state, $\rightarrow \subseteq Q \times A \times Q$ is the labeled transition relation. A *run* of $\mathcal{T}$ from q_1 to q_2 on a word $w \in A^*$ is denoted by $q_1 \xrightarrow{w}{}^* q_2$ and defined in the standard way. The language generated by $\mathcal{T}$, denoted $L(\mathcal{T})$, is defined to be $\{w \in A^* \mid q_0 \xrightarrow{w}{}^* q \text{ for some } q \in Q\}$. For a state $q \in Q$, we denote the language generated by $\mathcal{T}$ starting at q by $L_q(T)$ and define it to be $\{w \in A^* \mid q \xrightarrow{w}{}^* r \text{ for some } r \in Q\}$.

A *message sequence chart* (*MSC*) is a tuple $M = \langle P, E, C, \lambda, B, \{<_p\}_{p \in P}\rangle$ where:

- P is a finite set of processes, E is a finite set of events, and C is a finite set of message labels.

  The set of actions of M is defined to be $\Sigma_M = P \times \{!, ?\} \times P \times C$ where $(p, !, q, m)$ signifies process p sending message m to q, while action $(p, ?, q, m)$ signifies p receiving message m from q. All actions of the form $(p, !, q, m)$ are called *send* actions, while actions of the form $(p, ?, q, m)$ are called *receive* actions.

  For a process $p \in P$, the set $\Sigma_p = \{a \in \Sigma_M \mid a = (p, !, q, m) \text{ or } a = (p, ?, q, m)\}$ where $q \in P$ and $m \in C$, is the set of all actions in which p participates.
- $\lambda : E \rightarrow \Sigma_M$ is the labeling function which maps events to actions. For a process $p \in P$, the set $E_p = \{e \mid \lambda(e) \in \Sigma_p\}$ are the events in which p participates.

  E is further partitioned into send events $S = \{e \,|\, \lambda(e) = (p, !, q, m),\ p, q \in P,\ m \in C\}$ and receive events $R = \{e \,|\, \lambda(e) = (p, ?, q, m),\ p, q \in P,\ m \in C\}$ respectively.
- $B : S \rightarrow R$ is a bijective map which maps each send event to its corresponding receive event. We require that if $\lambda(e) = (p, !, q, m)$ then $\lambda(B(e)) = (q, ?, p, m)$. We refer to B as the "matching receive" map.
- For each $p \in P$, $<_p$ is a strict total order on E_p. In addition, the matching receive map B induces a strict partial order $<_B$ on E, which says that a receive event has to be preceded by the corresponding send event, defined by $e <_B e'$ iff $B(e) = e'$.

  We define $<_M$ as the transitive closure, and $\leq_M$ as the reflexive transitive closure, of $(\bigcup_{p \in P} <_p) \cup <_B$ respectively. It is required that the relation $\leq_M$ must be a partial-order.

A *linearization* of the events in an MSC M as defined above is a sequence of events $w = e_1 e_2 \cdots e_n \in E^*$ containing all the events of E without repetitions, and respecting the partial order $\leq_M$ in the sense that for no $i < j \leq n$ do we have $e_j \leq_M e_i$. We denote the set of linearizations of M by $lin(M)$. We define the

event language of M, written $L^e(M)$, to be the set of all prefixes of linearizations of M. Thus, $L^e(M) = \{x \mid x \preceq y,\ y \in lin(M)\}$.

Following [2], we define a *cut* in an MSC M to be a subset c of the events E of M which is closed with respect to the partial order $\leq_M$: i.e. if $e \in c$ and $e' \leq_M e$, then $e' \in c$. Each prefix of a linearization of an MSC corresponds to a sequence of "incremental" cuts as described below:

Lemma 1. *Let M be an MSC with event set E. Then $w \in E^*$ is a prefix of some linearization of M if and only if there exists a sequence of cuts $\langle c_x \rangle_{x \preceq w}$ in M such that $c_\epsilon = \emptyset$, and for each $x \cdot e \preceq w$, we have $c_{x \cdot e} - c_x = \{e\}$.* □

We now define the notion of a message sequence graph. A *Message Sequence Graph* (MSG) is a vertex-labeled graph G of the form $\langle V, v_0, \Delta, \mathcal{M}, \mu \rangle$, where V is the set of vertices of the MSG, $v_0 \in V$ is the initial vertex, $\Delta \subseteq (V \times V)$ is the set of directed arcs, $\mathcal{M}$ is the set of MSC's associated with the MSG, and $\mu : V \to \mathcal{M}$ maps each vertex of G to one of the MSCs in $\mathcal{M}$. We assume that the MSC's in $\mathcal{M}$ are all over a common set of processes and labels, and also that the events across the MSC's in $\mathcal{M}$ are distinct. The set of events of G is denoted E^G, and is defined to be $\bigcup_{v \in V} E_{\mu(v)}$. We denote the set of events where process p participates by E^G_p, defined in a similar way as for a single MSC. The set of action labels for G is defined to be $\Sigma_G = \bigcup_{v \in V} \Sigma_{\mu(v)}$. Fig. 1 shows an example MSG.

A *path* in G is a sequence of vertices $v_1, \ldots, v_k$ $(k \geq 0)$ of G such that $(v_i, v_{i+1}) \in \Delta$ for each $i \in \{1, \ldots, k-1\}$. An *initial path* in G is a path beginning at v_0. We will use the convention that α, β, etc. denote paths in G, and u, v etc. denote vertices in G.

Let π be a non-empty path in an MSG G over a common set of processes P and labels C. For each vertex v in G, let each MSC $\mu(v)$ be $\langle P_v, E_v, C, \lambda_v, B_v, \{<^v_p\}_{p \in P} \rangle$. We define the *(weak) concatenation* of the MSCs in the path π to be the MSC $M_\pi = \langle P, E_\pi, C, \lambda_\pi, B_\pi, \{<^\pi_p\}_{p \in P} \rangle$ where:

- $E_\pi = \bigcup_{\rho v \preceq \pi} (E_v \times \{\rho v\})$.
- For each $\rho v \preceq \pi$, we define $\lambda_\pi(e, \rho v) = \lambda_v(e)$.
- For each $\rho v \preceq \pi$, and for all send events $e \in E_v$, we define $B_\pi(e, \rho v) = (B_v(e), \rho v)$.
- For each $p \in P$, $<^\pi_p$ is given as follows: Let $\rho v \preceq \pi$ and $\rho' v' \preceq \pi$ and $e \in E_v$ and $e' \in E_{v'}$. Then $(e, \rho v) <^\pi_p (e', \rho' v')$ iff e and e' are p-events and either $\rho v \prec \rho' v'$ or $\rho v = \rho' v'$ and $e <^v_p e'$.

The set of linearizations of G, denoted by $lin(G)$, is defined to be

$$\{e_1 \cdots e_n \mid \pi \text{ initial path in } G, (e_1, \rho_1) \cdots (e_n, \rho_n) \in lin(M_\pi)\}.$$

The *event language* of G, denoted $L^e(G)$, is defined to be $\{u \mid u \preceq v \text{ and } v \in lin(G)\}$. The *action language* of G is denoted $L^a(G)$, and defined to be

$$\{\lambda_{v_1}(e_1) \cdots \lambda_{v_n}(e_n) \mid e_1 \cdots e_n \in L^e(G),\ \text{each } e_i \in E_{v_i}\}.$$

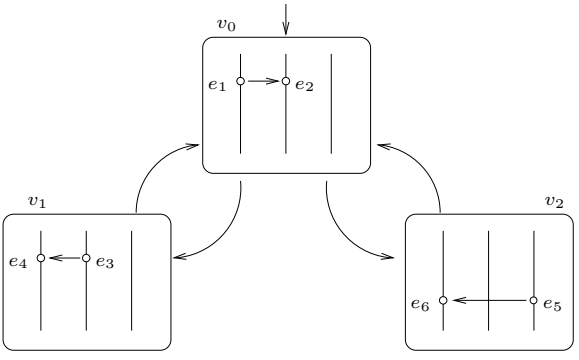


Fig. 1. An example MSG G_1

Finally, for an MSC $M = \langle P, E, C, \lambda, B, \{<_p\}_{p \in P}\rangle$, the *communication graph* of M is the directed graph on the set of processes P of M, where we have an edge (p, q) iff process p sends a message to process q in M (i.e. there exists an event $e \in E_p$ and an event e' in E_q, with $B(e) = e'$). The MSC M is said to be *com-connected* [2] if the communication graph of M contains at most one non-trivial strongly connected component and isolated nodes corresponding to processes not reacting in M. We say an MSG G is *com-connected* if for every loop, i.e. a path of the form $u\alpha u$, in G, the communication graph of $M_{u\alpha}$ is com-connected. We illustrate this using the MSG G_1 in Fig. 1. The communication graph for the loop $v_0v_1v_0$ is com-connected, whereas the loop $v_0v_2v_0$ is not com-connected. Thus the MSG G_1 is not com-connected.

Our aim in this paper is to provide a constructive proof of the following claim:

Theorem 1 ([2,8]). *Let G be a com-connected MSG. Then $L^e(G)$ is regular (i.e. it can be generated by a finite-state transition system).*

We will prove this claim in Sec. 5. In Sec 3 we first show that for a general (not necessarily com-connected) MSG G we can associate a (possibly infinite state) transition system $\mathcal{T}_G$ which generates precisely the language $L^e(G)$. In Sec 4 we give a couple of rules by which we can reduce the state space of $\mathcal{T}_G$, while preserving its language, to obtain a transition system $\mathcal{T}'_G$. Finally in Sec. 5 we show that when the given MSG is com-connected, this reduced transition system $\mathcal{T}'_G$ will indeed have a finite number of states.

3 Transition System for an MSG

In this section we show how we can associate a transition system $\mathcal{T}_G$ with a given *general* MSG G, which generates exactly the same event language as G. This transition system will, in general, have an infinite number of states. For the rest of this section we fix an MSG $G = \langle V, v_0, \Delta, \mathcal{M}, \mu\rangle$.

We begin with some preliminary notions. We define a *configuration* of G to be a pair of the form (π, c) where π is a path in G and c is a cut in M_π. Configurations will play the role of states in $\mathcal{T}_G$. We can view a configuration (π, c) as a snapshot of the events each process has completed in its process line in the MSC M_π. Each process p can be viewed to be positioned at last event it has performed.

Next we introduce some notation regarding insertion and deletion in paths and cuts in G. For a path $\pi = \alpha\beta$ in G, by $\pi + \alpha/\gamma$ we denote the sequence of nodes $\alpha\gamma\beta$ in G. We note that $\pi + \alpha/\gamma$ may not be a path in G. Similarly, if $\pi = \alpha\beta\gamma$ is a path in G, then we define $\pi - \alpha/\beta$ to mean the sequence of nodes $\alpha\gamma$ in G.

Let (π, c) be a configuration of G, and let $\pi = \alpha\beta$. Then we define the set of events corresponding to c in $\pi + \alpha/\gamma$, denoted $c + \alpha/\gamma$, to be

$$\{(e, \rho) \in c \mid \rho \preceq \alpha\} \cup \{(e, \alpha\gamma\rho) \mid (e, \alpha\rho) \in c \text{ and } \rho \neq \epsilon\}.$$

Once again, it is not necessary that $c + \alpha/\gamma$ is a cut in $M_{\alpha\gamma\beta}$.

Similarly, if (π, c) is a configuration of G, and $\pi = \alpha\beta\gamma$, we define the set of events corresponding to c in $\pi - \alpha/\beta$, denoted $c - \alpha/\beta$ to be

$$\{(e, \rho) \in c \mid \rho \preceq \alpha\} \cup \{(e, \alpha\rho) \mid (e, \alpha\beta\rho) \in c,\ \rho \neq \epsilon\}.$$

Let (π, c) be a configuration of G, with $\pi = \alpha\beta\gamma$. We say that β is *completely traversed* in c if all the events in β are included in c, and all processes which react in β have their maximal event in c located in γ. Formally, we represent this as a predicate $Ex(\pi, \alpha, \beta, c)$ which is true iff the conditions below are true:

- $E_{\alpha\beta} - E_\alpha \subseteq c$, and
- For each $p \in P$, if $(e, \alpha\rho v) \in c$ with $e \in E_p$ and $\rho v \preceq \beta$, then there exists $e' \in E_p$ and $\rho' v' \preceq \gamma$ such that $(e', \alpha\beta\rho' v') \in c$.

Similarly, we say β is *completely unexecuted* in c if none of the events in β are included in c: i.e. $(E_{\alpha\beta} - E_\alpha) \cap c = \emptyset$.

Consider a configuration (π, c) of G. We now want to define a way of cutting out loops in π which are completely unexecuted in c. We say that $\alpha_1 u_1 \beta_1 u_1 \cdots \alpha_n u_n \beta_n u_n \gamma$ with $n \geq 0$ is an *unexecuted loop decomposition* of π wrt c, if the following holds: $\pi = \alpha_1 u_1 \beta_1 u_1 \cdots \alpha_n u_n \beta_n u_n \gamma$; each $\alpha_i u_i$ has no completely unexecuted loops; each $\beta_i u_i$ is completely unexecuted; and γ does not have any completely unexecuted loops. It is not difficult to see that if we remove an unexecuted loop from a configuration, we get another valid configuration. The unexecuted loop-free configuration of (π, c) corresponding to the decomposition $\alpha_1 u_1 \beta_1 u_1 \cdots \alpha_n u_n \beta_n u_n \gamma$ is obtained by cutting out each of the $\beta_i u_i$'s, and is defined to be:

$$(\alpha_1 u_1 \cdots \alpha_n u_n \gamma, (\cdots (c - \alpha_1 u_1/\beta_1 u_1) \cdots) \ -\ \alpha_1 u_1 \cdots \alpha_n u_n/\beta_n u_n).$$

There can be several unexecuted loop decompositions for a given configuration, and hence also several unexecuted loop-free configurations. We denote by $\lfloor (\pi, c) \rfloor_{ue}$ the set of all unexecuted loop-free configurations of (π, c).

We also define a (unique) *left-most maximal* unexecuted loop decomposition of a configuration (π, c). We define this to be $\alpha_1 u_1 \beta_1 u_1 \cdots \alpha_n u_n \beta_n u_n \gamma$ with $n \geq 0$ where:

- $\pi = \alpha_1 u_1 \beta_1 u_1 \cdots \alpha_n u_n \beta_n u_n \gamma$
- Each $\beta_i u_i$ is the left-most unexecuted loop in the segment $\alpha_i u_i \beta_i u_i \cdots \beta_n u_n \gamma$. That is, for each $\tau_1 v \tau_2$ such that $\tau_1 v \tau_2 = \alpha_i$, there is no $\tau_3 v \preceq u_i \beta_i u_i \cdots \alpha_n u_n \beta_n u_n \gamma$ such that $\tau_2 \tau_3 v$ is completely unexecuted.
- Each $\beta_i u_i$ is maximal: That is, no prefix τu_i of $\alpha_{i+1} u_{i+1} \beta_{i+1} u_{i+1} \cdots \alpha_n u_n \beta_n u_n \gamma$ is completely unexecuted.
- γ does not have any completely unexecuted loops.

Each $\alpha_i u_i$ marks the beginning of the i-th maximal loop which is completely unexecuted in c, starting from the left of π. It is easy to see that this decomposition is unique. We define the (unique) *left-most maximal* unexecuted loop configuration induced by (π, c), denoted $\lfloor (\pi, c) \rfloor_{lue}$, to be:

$$(\alpha_1 u_1 \cdots \alpha_n u_n \gamma, (\cdots (c - \alpha_1 u_1 / \beta_1 u_1) \cdots) \ - \ \alpha_1 u_1 \cdots \alpha_n u_n / \beta_n u_n).$$

We are now in a position to describe a transition system corresponding to the given MSG G. We define $\mathcal{T}_G = (Q, E_G, q_0, \rightarrow)$ where: Q is the set of configurations of G; E_G is the set of events of G as defined in the last section; the initial state is $q_0 = (\epsilon, \emptyset)$; and the transition relation $\rightarrow$ is given by the following rules:

(T1) $(\pi, c) \xrightarrow{e} (\pi, c')$ provided $c' = c \cup \{(e, \rho)\}$ for some $\rho \preceq \pi$ and $(e, \rho) \notin c$.

(T2) $(\pi, c) \xrightarrow{e} (\pi \rho v, c')$ provided $c' = c \cup \{(e, \pi \rho v)\}$ and ρv is loop-free. Also, if $\pi = \epsilon$ then ρv has to be an initial path in G.

(T3) $(\pi_1 u \pi_3, c) \xrightarrow{e} \lfloor (\pi_1 u \pi_2 u \pi_3, c') \rfloor_{lue}$ provided there exists a non-empty and loop-free α and a loop-free β such that $\pi_2 u = \alpha\beta$ and $c + \pi_1 u / \pi_2 u$ is a cut in $M_{\pi_1 u \pi_2 u \pi_3}$ and c' is $(c + \pi_1 u / \pi_2 u) \cup \{(e, \pi_1 u \alpha)\}$.

We illustrate these transition rules in Fig. 2. We note that all configurations (π, c) of $\mathcal{T}_G$ reachable from the initial state, satisfy that properties that they always have a process in the last node of π, and also that they are always unexecuted loop free.

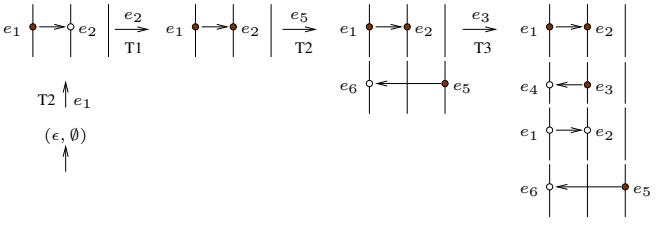


Fig. 2. Initial transitions in $\mathcal{T}_G$ for MSG G_1.

We now sketch a proof of the correctness of our construction of $\mathcal{T}_G$, by showing that it accepts exactly the language $L^e(G)$. We first show that $\mathcal{T}_G$ is "complete" in the sense that it accepts all event sequences in $L^e(G)$. Let $w \in L^e(G)$ with $w = e_1 e_2 \cdots e_n$. Then we know by Lemma 1 that there is a sequence of incremental cuts $c_0, c_1, \cdots, c_n$ in M_π for some initial path π in G, such that $c_0 = \emptyset$ and for each $i \in \{0, \ldots, n-1\}$, $c_{i+1} - c_i = \{(e_{i+1}, \rho_{i+1})\}$ for some $\rho_{i+1} \preceq \pi$. For each $i \in \{0, \ldots, n\}$, let θ_i be $\max\{\rho_j \mid j \leq i\}$. We claim that w has a run

$$(\pi_0, d_0) \xrightarrow{e_1} (\pi_1, d_1) \xrightarrow{e_2} \cdots \xrightarrow{e_n} (\pi_n, d_n)$$

in $\mathcal{T}_G$, where $\pi_0 = \epsilon$, $d_0 = \emptyset$, and for all $j \in \{0, \ldots, n\}$, $(\pi_j, d_j) \in \lfloor (\theta_j, c_j) \rfloor_{ue}$.

We do this by showing, using induction on i, that for each $i \in \{0, \ldots, n\}$ we can produce a run

$$(\pi_0, d_0) \xrightarrow{e_1} (\pi_1, d_1) \xrightarrow{e_2} \cdots \xrightarrow{e_i} (\pi_i, d_i)$$

in $\mathcal{T}_G$ such that $(\pi_0, d_0) = (\epsilon, \emptyset)$, and for each $j \in \{1, \ldots, i\}$, $(\pi_j, d_j) \in \lfloor (\theta_j, c_j) \rfloor_{ue}$. The proof is fairly routine and the complete details can be found in [3].

We now argue the "soundness" of $\mathcal{T}_G$, by showing that if $w \in L(\mathcal{T}_G)$ then $w \in L^e(G)$. Let $w \in L(\mathcal{T}_G)$ with $w = e_1 e_2 \cdots e_n$ such that $(\pi_0, c_0) \xrightarrow{e_1} (\pi_1, c_1) \cdots \xrightarrow{e_n} (\pi_n, c_n)$ is a run in $\mathcal{T}_G$. We note that each π_i must be an initial path in G. We claim that w will produce a sequence of incremental cuts $c'_0, c'_1, \ldots, c'_n$ in M_{π_n} where $c'_0 = \emptyset$, $c'_n = c_n$, and for each $i \in \{0, \ldots, n-1\}$, $c'_{i+1} - c'_i = \{(e_{i+1}, \rho_{i+1})\}$ for some $\rho_{i+1} \preceq \pi_n$. We prove this by induction on length of w, and again the details can be found in [3].

4 Reducing $\mathcal{T}_G$

In this section our aim is to show that the state-space of the transition system $\mathcal{T}_G$ can be reduced, by observing that we can remove fully traversed prefixes and loops from configurations without affecting the language generated by the transition system. To do this it will be convenient to make use of the notion of bisimilarity and some simple results concerning it.

Let $\mathcal{T} = (Q, A, q_0, \rightarrow)$ be a transition system and let $\mathcal{R}$ be a bisimulation relation on $\mathcal{T}$. We represent the reflexive transitive closure of $\mathcal{R}$ as $\mathcal{R}^*$ and clearly, it is also a bisimilar relation.

The lemma below shows how we can reduce the state space of a transition system using a bisimulation relation on it.

Lemma 2. *Let $T = (Q, A, q_0, \rightarrow)$ be a transition system, and let $\mathcal{R}$ be bisimulation relation on $\mathcal{T}$. Consider a transition system $\mathcal{T}' = (Q', A, q_0, \Rightarrow)$ where $Q' \subseteq Q$, and $\Rightarrow$ satisfies the following conditions for any $q' \in Q$:*

- *whenever $q' \xrightarrow{a} r$ in $\mathcal{T}$, there exists a state $r' \in Q'$ such that $q' \overset{a}{\Rightarrow} r'$ in $\mathcal{T}'$, and $(r, r') \in \mathcal{R}$.*
- *whenever $q' \overset{a}{\Rightarrow} r'$ in $\mathcal{T}'$, there exists $r \in Q$ such that $q' \xrightarrow{a} r$ in $\mathcal{T}$ and $(r, r') \in \mathcal{R}$.*

Then $L(\mathcal{T}) = L(\mathcal{T}')$. □

We return now to our transition system $\mathcal{T}_G$ corresponding to the given MSG G. Consider the relation $\rightsquigarrow_1$ on the states of $\mathcal{T}_G$ defined below, which relates a configuration with the one obtained from it by deleting a fully traversed prefix. We define $\rightsquigarrow_1$ as follows: $(\alpha\beta, c) \rightsquigarrow_1 (\beta, c')$ provided

- α is non-empty,
- $Ex(\alpha\beta, \epsilon, \alpha, c)$ holds, and
- $c' = c - \epsilon/\alpha$.

The relation $\rightsquigarrow_2$ relates configurations of $\mathcal{T}_G$ based on deleting completely traversed loops. We have $(\alpha u\beta u\gamma, c) \rightsquigarrow_2 (\alpha u\gamma, c')$ provided

- $Ex(\alpha u\beta u\gamma, \alpha, u\beta, c)$ holds, and
- $c' = c - \alpha/u\beta$.

Lemma 3. *The relations $\rightsquigarrow_1$ and $\rightsquigarrow_2$ are bimulation relations on the transition system $\mathcal{T}_G$.* □

We will now reduce the state space of $\mathcal{T}_G$ using the bisimulation relations defined above.

For a configuration (π, c) of G, we define its *maximal reducible decomposition* to be $\alpha\alpha_1u_1\beta_1u_1 \cdots \alpha_nu_n\beta_nu_n\gamma$ with $n \geq 0$, where

- $\pi = \alpha\alpha_1u_1\beta_1u_1 \cdots \alpha_nu_n\beta_nu_n\gamma$.
- α is the maximal prefix of π which is completely traversed.
- Each $u_i\beta_i$ is the leftmost completely traversed loop in the segment $\alpha_iu_i\beta_iu_i \cdots \alpha_nu_n\beta_nu_n\gamma$.
- Each $u_i\beta_i$ is maximal.
- γ does not have any completely traversed loop.

The maximal reducible decomposition of (π, c) can be seen to be unique.

Let $\alpha\alpha_1u_1\beta_1u_1 \cdots \alpha_nu_n\beta_nu_n\gamma$ be the maximal reducible decomposition of (π, c). Then we define the *reduced form* of (π, c), denoted $\lfloor(\pi, c)\rfloor$, to be the configuration

$$(\alpha_1u_1 \cdots \alpha_nu_n\gamma, \\ (\cdots((c - \epsilon/\alpha) - \alpha_1u_1\beta_1) - \cdots) - \alpha_1u_1 \cdots \alpha_nu_n/u_n\beta_n).$$

Clearly, the reduced form of a configuration does not contain any completely traversed prefix or loops.

We now define the reduced transition system corresponding to the MSG G, denoted $\mathcal{T}'_G$, as follows: $\mathcal{T}'_G = (Q', E_G, q_0, \Rightarrow)$ where

- $Q' = \{\lfloor(\pi, c)\rfloor \mid (\pi, c)$ a configuration of $G\}$.
- E_G is the set of events of G.
- $q_0 = (\epsilon, \emptyset)$.
- Let $q', r' \in Q'$. Then, $q' \stackrel{e}{\Rightarrow} r'$ iff there exists $r \in Q$ such that $q' \stackrel{e}{\rightarrow} r$ in $\mathcal{T}_G$, and $r' = \lfloor r\rfloor$.

We note that the reduced form of the initial configuration $(\epsilon, \emptyset)$ is itself.

We now want to argue that $\mathcal{T}'_G$ generates the same language as $\mathcal{T}_G$. We have already shown that $\rightsquigarrow_1$ and $\rightsquigarrow_2$ are bisimulation relations on $\mathcal{T}_G$. Hence so is the relation $(\rightsquigarrow_1 \cup \rightsquigarrow_2)^*$. Further, for any configuration (π, c), it is clear that $((\pi, c), \lfloor(\pi, c)\rfloor) \in (\rightsquigarrow_1 \cup \rightsquigarrow_2)^*$. Finally, it is immediate to check that T'_G satisfies the conditions of Lemma 2 with respect to $\mathcal{T}_G$ and the bisimulation relation $\mathcal{R} = (\rightsquigarrow_1 \cup \rightsquigarrow_2)^*$. Hence by Lemma 2 we can conclude that $L(\mathcal{T}'_G) = L(\mathcal{T}_G)$.

We summarise the result of this section in the following theorem:

Theorem 2. *For any MSG G, the reduced transition system $\mathcal{T}'_G$ generates precisely the language $L^e(G)$.*

We note that though $\mathcal{T}'_G$ has fewer states than $\mathcal{T}_G$ in general, it may still have an infinite number of states. Consider the MSG G_2 in Fig. 3. The configurations of the form $(v_0^n, \{(e_1, v_0), (e_1, v_0^2), \cdots, (e_1, v_0^n)\})$ where $n > 0$ do not have any unexecuted loops or completely traversed loops. So, for each n, these are distinct states in $\mathcal{T}'_{G_2}$, and thus $\mathcal{T}'_{G_2}$ has an infinite number of states. In the next section we show that this is not possible for a com-connected MSG.

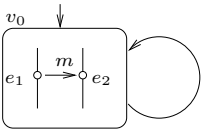


Fig. 3. MSG G_2 with an infinite-state $\mathcal{T}'_{G_2}$

5 Regularity of Com-Connected MSG's

In this section our aim is to supply a proof of Theorem 1. We begin with an observation from [2].

Lemma 4 ([2]). *Let G be a com-connected MSG. Consider a configuration of the form $(\alpha u \beta u \gamma, c)$. Then, either $u\beta$ is completely unexecuted, or $u\beta$ is completely traversed, or there is a process whose last executed event and next unexecuted event are both in $u\beta$.*

Proof. Let us assume the contrary. Then among the processes that take part in an event in the nodes in $u\beta$, there must be processes p and q such that *none* of the p events in $u\beta$ are executed, and *all* of the q events in $u\beta$ are executed. As G is com-connected, we must have a path in the communication graph of $M_{u\beta}$ from p to q. Let this path be $p = r_0 \rightarrow r_1 \rightarrow \cdots \rightarrow r_n = q$ with $n \geq 1$. Then, since q has completed all its events in $u\beta$, it must have also received a message from r_{n-1}. Thus r_{n-1} has taken part in $u\beta$, since it must have sent the message received by q. Now either r_{n-1} has another event to take part in $u\beta$, and we are done; or, it has completed all its events in $u\beta$ and in particular has received a message from r_{n-2}. We repeat this argument till we either find a process r_i

which has participated in $u\beta$ and has an unexecuted event in $u\beta$, or reach a contradiction that process p has participated in an event in $u\beta$. This completes the proof of the lemma. □

Let us now consider the reduced transition system $\mathcal{T}'_G$ for a com-connected MSG G. The states in $\mathcal{T}'_G$ are all in reduced form, and hence have no completely unexecuted loops or completely traversed loops. It follows that in any loop in a state (π, c) of $\mathcal{T}'_G$, there must be at least one process positioned in a node in that loop. Thus no node can occur more than $k+1$ times in π, where k is the number of processes. In fact, because of the property of the reachable configurations of $\mathcal{T}_G$ that there is always a process positioned in the last node of the path, the bound of $k+1$ can be reduced to k. This bounds the length of π by mk, where m is the number of nodes in G. We also get an upper bound of $m^{mk}(mnk)^k$ on the number of states in $\mathcal{T}'_G$, where n is the maximum number of events in any MSC associated with a node in G.

This completes the proof of Theorem 1.

6 Synchronous MSG's

In this section we consider MSC's with synchronous (or "handshake") messages. These synchronous MSC's are defined in a similar manner to the (asynchronous) MSC's of Sec. 2, except that sends and receives are no longer distinct events, but are represented as a single message exchange event. We call MSG's whose nodes are labelled by synchronous MSC's *synchronous* MSG's. The language of event sequences generated by a synchronous MSG is defined in a similar manner to that of (asynchronous) MSG's in Sec. 2. Finally, we say a synchronous MSG G is *com-connected* iff for every loop $u\beta u$ in G, the communication graph of $M_{u\beta}$ has at most one non-trivial connected component. Fig. 4 shows a com-connected synchronous MSG G_4.

The construction of the transition systems $\mathcal{T}_G$ and $\mathcal{T}'_G$ for an asynchronous MSG G, is based purely on the partial order induced by a path in the MSG. Hence it readily applies to synchronous MSG's as well. Further, the proof of

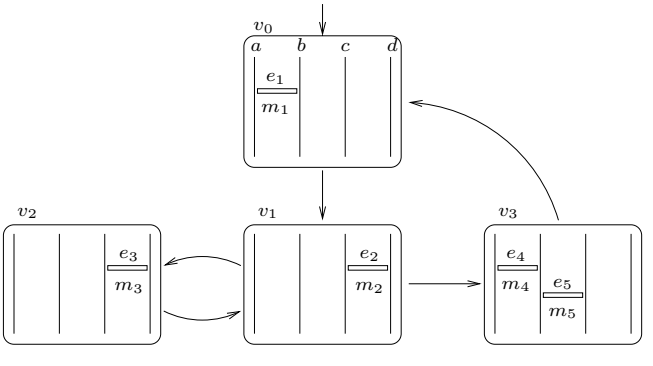


Fig. 4. MSG G_4 with synchronous messages

correctness of $\mathcal{T}_G$ and $\mathcal{T}'_G$ also makes use of certain properties of this partial order, and these properties are easily seen to be satisfied by synchronous MSG's as well. Hence we can conclude:

Theorem 3. *Let G be a synchronous MSG, and let $\mathcal{T}_G$ and $\mathcal{T}'_G$ be the transition systems defined in Sec. 3 and 4 respectively. Then both transition systems generate exactly the language $L^e(G)$.*

Further, Lemma 4 can also be seen to apply for synchronous MSG's, and hence we have:

Theorem 4. *Let G be a com-connected synchronous MSG. Then $L^e(G)$ is regular.*

The state transition diagram of the transition system T'_{G_4} for the MSG G_4 of Fig. 4 can be found in [3].

The MSG of Fig. 4 is useful for pointing out the incompleteness of the transition system constructed by Uchitel in [11]. He constructs a transition system, called the "trace model," which is meant to generate the language of event sequences for a given com-connected synchronous MSG. The construction uses a "coordinator" component which keeps track of the current path in the MSG traced out by processes, and ensures that other processes follow this path consistently. The transition rules are similar to ours, except that there is *no rule* for inserting a path within the current node list. This ommission essentially leads to the trace model being incomplete.

The trace model generated by the LTSA-MSC tool [10,12] on the MSG G_4 as input is shown in Fig. 5. The labels of the trace model basically corresponds

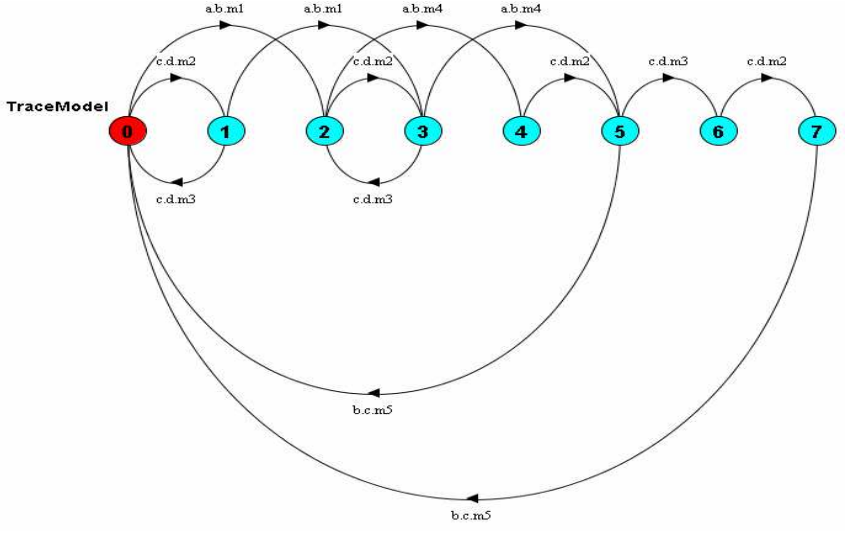


Fig. 5. Transition system generated by LTSA-MSC tool for MSG G_4

to the events of the MSG. For example, the transition from state 0 to state 2 happens with the synchronous exchange of m_1 between process a and b, which is the event e_1 in the MSG G_4. The event sequence $e_1e_4e_2e_3e_2e_3$ is not allowed by the trace model, while it is clearly a legal behaviour of the MSG G_4, being a prefix of a linearisation of M_π, where π is the path $v_0v_1v_2v_1v_2v_1v_3$.

7 Detecting Implied Scenarios

We now sketch how our construction of $\mathcal{T}'_G$ can be used to detect "implied scenarios" in a com-connected synchronous MSG.

A synchronous MSG G induces a natural "minimal" distributed finite-state system model (called the "architecture model" in [11]). Each process has a component obtained by keeping track of which events it can participate in next. The system model induced by the components, can be viewed as the "synchronised product" of the components for each process, where a pair of components are required to simultaneously execute events that are common to their process lines. The system model, which we denote $\mathcal{S}_G$, can be seen to include *all* the behaviours in $L^e(G)$ for a given synchronous MSG G. The problem is that it may sometimes generate a strict superset of the behaviours in $L^e(G)$, and these behaviours are what are refered to as "implied scenarios." Thus an event sequence $w \in E_G^*$ is called an *implied scenario* if $w \in L(\mathcal{S}_G) - L^e(G)$. We refer the reader to [13,11] for some illustrative examples of implied scenarios.

For a com-connected synchronous MSG G implied scenarios can be easily detected from $\mathcal{T}'_G$ and $\mathcal{S}_G$ using standard automata-theoretic techniques. The MSG G_4 can be seen to have no implied scenarios [3]. But, since the trace model generated by the LTSA-MSC tool [10,11,12] is incomplete, it reports $e_1e_4e_2e_3e_2e_3$ as one of the several implied scenarios for this MSG.

8 Conclusion

In this paper we have given a precise construction of a transition system for a given MSG specification, which accepts exactly the set of behaviours specified by the MSG. When the given MSG specification is com-connected, the transition system we give is guaranteed to be finite-state and can thus be used a basis for building sound and complete tools for analysing properties of these specifications. Our transition system is also suitable for analysing MSG specifications in a bounded fashion, even when the given MSG is not com-connected.

References

1. Alur, R., Etessami, K., Yannakakis, M.: Inference of message sequence charts. IEEE Trans. Software Eng. 29(7), 623–633 (2003)
2. Alur, R., Yannakakis, M.: Model checking of message sequence charts. In: Baeten, J.C.M., Mauw, S. (eds.) CONCUR 1999. LNCS, vol. 1664, pp. 114–129. Springer, Heidelberg (1999)

3. Chakraborty, J., D'Souza, D., Narayan Kumar, K.: Analysing Message Sequence Graph Specifications, Technical Report IISc-CSA-TR-2009-1, CSA Department, IISc (2009), http://www.csa.iisc.ernet.in/TR/2009/1/
4. Clerbout, M., Latteux, M.: Semi-commutations. Inf. Comput. 73(1), 59–74 (1987)
5. Genest, B., Muscholl, A., Peled, D.: Message sequence charts. In: Desel, J., Reisig, W., Rozenberg, G. (eds.) Lectures on Concurrency and Petri Nets. LNCS, vol. 3098, pp. 537–558. Springer, Heidelberg (2004)
6. Henriksen, J.G., Mukund, M., Kumar, K.N., Sohoni, M.A., Thiagarajan, P.S.: A theory of regular msc languages. Inf. Comput. 202(1), 1–38 (2005)
7. Muccini, H.: Detecting implied scenarios analyzing non-local branching choices. In: Pezzé, M. (ed.) FASE 2003. LNCS, vol. 2621, pp. 372–386. Springer, Heidelberg (2003)
8. Muscholl, A., Peled, D.: Message sequence graphs and decision problems on mazurkiewicz traces. In: Kutyłowski, M., Wierzbicki, T., Pacholski, L. (eds.) MFCS 1999. LNCS, vol. 1672, pp. 81–91. Springer, Heidelberg (1999)
9. Muscholl, A., Petersen, H.: A note on the commutative closure of star-free languages. Inf. Process. Lett. 57(2), 71–74 (1996)
10. Uchitel, S.: LTSA-MSC tool (2001), http://www.doc.ic.ac.uk/~su2/Synthesis/
11. Uchitel, S.: Incremental Elaboration of Scenario Based Specifications and Behavior Models Using Implied Scenarios. PhD thesis, Imperial College (2003)
12. Uchitel, S., Chatley, R., Kramer, J., Magee, J.: LTSA-MSC: Tool support for behaviour model elaboration using implied scenarios. In: Garavel, H., Hatcliff, J. (eds.) TACAS 2003. LNCS, vol. 2619, pp. 597–601. Springer, Heidelberg (2003)
13. Uchitel, S., Kramer, J., Magee, J.: Detecting implied scenarios in message sequence chart specifications. In: ESEC / SIGSOFT FSE, pp. 74–82 (2001)
14. Uchitel, S., Kramer, J., Magee, J.: Incremental Elaboration of Scenario-Based Specifications and Behavior Models Using Implied Scenarios. ACM Transactions on Software Engineering and Methodology 13(1), 37–85 (2004)

Optimize Context-Sensitive Andersen-Style Points-To Analysis by Method Summarization and Cycle-Elimination

Li Qian[1,2], Zhao Jianhua[1,2], and Li Xuandong[1,2]

[1] The State Key Laboratory of Novel Software Technology of China
[2] Department of Computer Science and Technology, Nanjing University
210093, Nanjing, P.R. China
qianjie@seg.nju.edu.cn, {zhaojh,lxd}@nju.edu.cn

Abstract. This paper presents an efficient context-sensitive, field-based Andersen-style points-to analysis algorithm for Java programs. This algorithm first summarizes methods of the program under analysis using directed graphs. Then it performs local circle elimination on these summary graphs to reduce their sizes. The main analysis algorithm uses these graphs to construct the main points-to graph. Topological sort and cycle-elimination is performed on the nodes of both main points-to graphs and summary graphs to speed up the transitive closure computation on the main points-to graph. A suite of Java program benchmarks are used to demonstrate the efficiency of our algorithm.

1 Introduction

Points-to analysis determines the set of objects that a reference variable or a reference object-field may point to. This information is important to many applications in program understanding and verification. The precision and performance of these applications depend heavily on the precision of the points-to analysis algorithm. For example, using precise points-to information, a program verification tool for detecting race condition can avoid many false warnings. So, precise and efficient points-to analysis is an important infrastructure for automatic program verification/understanding tools.

Unfortunately, precise pointer analysis is NP-hard [10] [11]. So people usually make trade-offs between precision and efficiency. The factors that affect points-to analysis precision directly are [3].

- **Flow sensitivity.** Flow-sensitive analysis takes the execution order of program statements into account. It usually performs strong updates, i.e. an assignment to a variable u makes u point to the new object, and stops u from pointing to its original object. A flow-insensitive analysis ignores the execution order and assumes that u may point to both objects. Thus, flow-sensitive algorithms are more precise but more expensive than flow-insensitive ones.
- **Context sensitivity.** Informally speaking, if an analysis distinguishes between different calling-contexts of a method, it is context-sensitive; otherwise, the analysis is called context-insensitive. In a context-sensitive analysis, a local variable may have different points-to information in different calling-contexts of the method, while a context insensitive analysis neglects the information of calling-contexts.

T. Margaria and B. Steffen (Eds.): ISoLA 2010, Part I, LNCS 6415, pp. 564–578, 2010.

- **Object representation.** Objects in program can be treated differently. If all instances of a class are represented by one abstract object, i.e. all of them are treated as a whole, the analysis is called object-insensitive. If one abstract object is created for each object creation site respectively, i.e. the objects created by same creation site are treated as a whole, the analysis is called object-sensitive. If the algorithm adds context information on creation site, it is called heap-clone.
- **Field sensitivity and field-based.** Field sensitive analysis represents the fields of an object distinctly. If the fields in an object are indistinguishable with respect to what they reference, the analysis is termed field-insensitive. If the fields in an object are distinguished without context information, the analysis is called field-based.

Andersen's points-to analysis [1] is an inclusion-based points-to analysis algorithm. It is considered as the most precise flow-insensitive and context-insensitive points-to analysis. This algorithm is closely related to computing the transitive closure of a directed graph. Inclusion constraints between program variables are first generated through a source code analysis. A directed graph is constructed to represents all such constraints: nodes are used to represent variables, and directed edges represent the inclusion constrains between variables. Then points-to information is gathered by computing the transitive closure of this graph. The transitive closure of the final graph yields the points-to solution.

The context-insensitive nature of Andersen's algorithm is a major reason for the imprecision of analysis results: though local variables and arguments in a method may point to different objects when the method is invoked in different calling-contexts, Andersen's algorithm ignores context information, and computes the same points-to information for different invocations of a method.

Andersen's algorithm has a complexity of $\mathcal{O}(n^3)$, where n is the number of nodes in the graph, i.e. the number of variables and objects. Reducing the input size n is the key factor of the algorithm's scalability. On the other hand, a context-sensitive extension of the Andersen's algorithm has to deal with many more nodes, since a local variable may have many nodes in the graph corresponding to different calling-contexts.

This paper presents a scalable context-sensitive, field-based extension of Andersen's algorithm for Java programs. The algorithm consists of two phases: the method-summarization phase and the inter-procedure analysis phase. In the method summarization phase, this algorithm summarizes each method in the Java program under analysis using a directed graph; then in the inter-procedure analysis phase, it iteratively uses these method summaries to compute the main points-to graph. Our algorithm significantly reduces the node count of the main points-to graph by reducing the sizes to method summaries before the second phase. We also speed-up the second phase by sorting the nodes in the main points-to graph. We have implemented this analysis algorithm and evaluated its precision and efficiency by a set of Java benchmark programs.

The rest of this paper is organized as follows. In section 2 we give a brief introduction about the method summarization approach used in our algorithm. In section 3 we describe the main framework of our algorithm. Section 4 introduces a novel approach to topologically sort the graph nodes and eliminate the cycles in the graph. This approach both reduces the size of the graph and speeds up the transitive-closure computation. Two optimizations on summary graphs are also presented in this section. The

following section presents our experimental evaluation and section 6 compares our work with related works. A conclusion is given in the final section.

2 Method Summarization

Rather than operates on the source code directly, our algorithm first summarizes each method in the program under analysis, and then uses these summaries to compute points-to information. This approach has two advantages. First, the inter-procedure algorithm does not have to parse source code repeatedly, thus improving the analysis efficiency and the scalability of the analysis. Second, these method summaries can be pre-treated to reduce the size of the main points-to graph, which will be constructed in the inter-procedure phase.

2.1 Atomic Statements

To summarize a method, program statements that effect the set of objects pointed to by variables, called *points-to set* of these variables, should be analyzed. Such statements include all assignments performed by the method, all method invocations in the method, `return` statements, and actual-parameter/formal-parameter bindings.

Our algorithm first transforms each method into a list of *atomic statements*. Each atomic statement has one of the following forms.

- Copy statements: $l = r$
- Object creation statements: $l = new\ C()$
- Field read statements: $l = r.f$
- Field write statements: $l.f = r$
- Method invocation statements: $l = r_0.m(r_1, \ldots, r_k)$
- Return statements: `return` l

Here, l, r, r_i are program variables or temporary variables; f is the field (i.e. member variable) name; C are class names; m is the method name.

The transformation function τ which breaks a program statement into a set of atomic statements is defined as follow. Here, $t, t_0, t_1, \ldots$ are newly introduced temporary variables during the transformation process; $exp, exp_1, \ldots$ are expressions.

- $\tau(s) = \{s\}$ if the program statement is already an atomic one.
- $\tau(exp_1 = exp_2.b.f) = \tau(t = exp_2.b) \cup \tau\{exp_1 = t.f\}$
- $\tau(exp_1 = exp_2[exp_3]) = \tau(exp_3) \cup \tau(t = exp_2) \cup \tau(exp_1 = t.farr)$.
- $\tau(exp_1.f = exp_2) = \tau(t = exp_1) \cup \tau(t.f = exp_2)$
- $\tau(exp_1[exp_2] = exp_3) = \tau(exp_2) \cup \tau(t_1 = exp_1) \cup \tau(t_3 = exp_3) \cup \{t_1.farr = t_3\}$.
- $\tau(v = exp_0.m(exp_1, \ldots, exp_k)) = \tau(t_0 = exp_0) \cup \tau(t_1 = exp_1) \cup \ldots \cup \tau(t_k = exp_k) \cup \{v = t_0.m(t_1, \ldots, t_k)\}$.
- $\tau($`return` $exp) = \tau(t = exp) \cup \{$`return` $t\}$
- $\tau(exp =$ `new` $C(exp_1, \ldots, exp_k)) = \tau(t = exp) \cup \{t =$ `new` $C()\} \cup$ $\tau(t.$`initialM`$(exp_1, \ldots, exp_k))$, where `initialM` corresponds to the constructor of C.

In our algorithm, all the elements in an array are treated as a single field $farr$ of the array variable. So an array element reference is treated as a field-write/read operation to this array variable. The array index expressions in the definitions of $\tau(exp_1[exp_2] = exp_3)$ and $\tau(exp_1 = exp_2[exp_3])$ are further transformed by τ. The values of these expressions are ignored. The function `initialM` in the definition of $\tau(exp = \texttt{new}\ \ C(exp_1, \ldots, exp_k))$ is chosen from the constructor list of the class C, with respect to the given argument list. It initializes the fields of the object created by the statement.

The definitions shown above introduce a lot of temporary variables. These temporary nodes will be a huge burden in the second phase of our points-to algorithm. However, many of them can be removed using the approach in section 4.1.

2.2 Method Summary

A summary of a given method describes the digested information of this method. Formally speaking, a method summary of m consists of two elements (C, G): C describes the set of call sites in m, and G is a graph representing the effect of the atomic statements on points-to sets.

- C: a set of call site summaries. A call site summary is a tuple $< m, rec, ret, P >$, where m is the signature of this call site, rec is the call receiver, ret is the node to which the return value is passed to (ret is `null` if the invocation statement does not pass the returned value to any variables.) and P is the set of real parameters.
- G is a graph representing the effect of the atomic statement set of this method.
  - The node set N of G is a union of two disjoint sets, N_v and N_o
    - N_v: a set of nodes representing local variables. Our algorithm also treats some other values as local variables: formal parameters have corresponding nodes in N_v; a node labeled '`this`' in N_v represent the variable `this`; a node labeled '`ret`' in N_v represents the value returned by the method. The static variables are modeled as the fields of a special abstract object labeled '`static`'.
    - N_o: a set of nodes represent the objects created by the method.
  - The edge set E of G has four kinds of edges.
    - *points-to* edges, representing points-to relationship between variable nodes in N_v and object nodes in N_o;
    - *inclusion* edges, which say that the points-to set of one node in N_v includes that of another nodes in N_v;
    - *field-read* edges, corresponding to atomic statements of the form $l = r.f$;
    - *field-write* edges, corresponding to atomic statements of the form $l.f = r$.

Information about call sites can be easily derived by analyzing the atomic statement set of the method. To construct the summary graph G, the algorithm iterates over the atomic statement set and adds edges into G according to the rules depicted in Fig 1.

The example depicted in Fig 2 illustrates how to construct a summary graph of a method. The left-side shows a method $changeYF()$ of a class X. First, the statement $this.y = nx.y.setF(newf)$ in line (5) is broken into three atomic statements:

Atomic statement	Edge added
$l = \texttt{new } C()$	points-to edge from n_l to n_o corresponding to this instance create
$l = r$	inclusion edge from n_l to n_r
$l.f = r$	field-write edge from n_l to n_r labeled f
$l = r.f$	field-read edge from n_l to n_r labeled f
`return` l	inclusion edge from n_l to the node labeled 'ret'

Fig. 1. Points-to effect of atomic statements

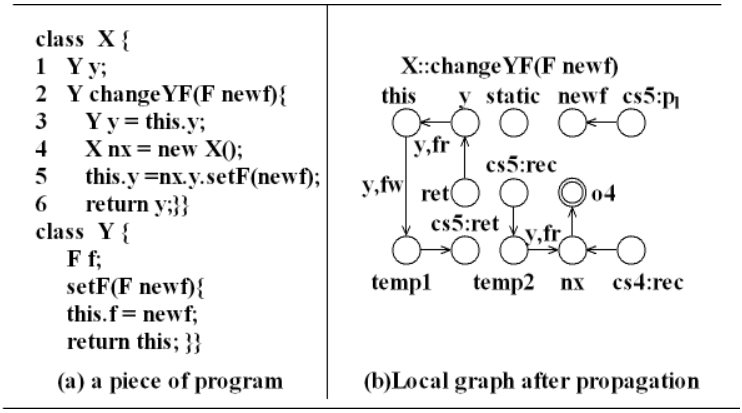


Fig. 2. Method points-to set representation

$this.y = temp1$, $temp2 = nx.y$ and $temp1 = temp2.setF(newf)$. The statement $nx = newX()$; is broken into $nx = newX()$; and $nx.initX()$;

There are two call-sites in $changeYF()$. $cs4 = < initX, cs4 : rec, \texttt{null}, \emptyset >$ is a special call-site derived from the object creation statement in line (4). Notice that $initX$ has no return value or formal parameters, and nx is treated as the receiver node of this call site. So only the node $cs4 : rec$ represents the object receiving this invocation is added into the graph. As to call site $cs5 = < setF, cs5 : rec, cs5 : ret, \{cs5 : p_1\} >$ in line (5), we add three more nodes into the graph: $cs5 : ret$ represents the value return by this call-site; $cs5 : rec$ represents the object receiving this invocation; $cs : p_1$ represents the formal parameter of this call-site.

Other nodes in the graph are nx, y, `this`, `ret`, $newf$, $temp1$, and $temp2$, which represent the local variable nx and y, the value `this`, the return value of $changeYF$, the formal parameter $newf$, the temporary variables $temp1$ and $temp2$, respectively.

We use the node labeled $o4$ to denote the objects created by the statement in line (4). This is the only node in N_o of this summary graph.

For statement 3, a field-read edge labeled y from node y to node `this` is added. A points-to edge from node nx to node $o4$ is added because of the statement at line (4). For statement in line (6), a inclusion edge from node `ret` to node y is added. The statement in line (5) is broken into three atomic statements as described above. A set of edges are added: a field-read edge from $temp2$ to nx labeled y and three inclusion edges: from $temp1$ to $cs5 : ret$, from `cs5:rec` to $temp2$, from $cs5 : p_1$ to $newf$.

Table 1. The Framework of The Inter-Procedure Analysis

INPUT: Summary graphs of all methods
OUTPUT: The main points-to graph
VARIABLE: $worklist$

POINTSTOANALYSIS ( )
{
  Add the initial calling-context $ctxt_0$ into $worklist$;
    /*$ctxt_0$ is a virtual calling-context with only one call-site invoking `main()`*/
  while($worklist$ is not empty)
  {
    while($worklist$ is not empty){
      $ctxt$:= a calling-context in $worklist$;
      $worklist$:= $worklist$ - $\{callsite\}$;
      Expanding $ctxt$ as described in subsection 3.3
    }
    Computing the transitive closure and adding new calling-contexts
        to $worklist$ as described in subsection 3.4
  }
}

3 The Framework of the Inter-Procedure Analysis

Our algorithm is a context-sensitive, field-based extension of Andersen's points-to analysis algorithm. Our algorithm constructs the main points-to graph dynamically. It starts from the `main()` method, then iteratively expands calling-contexts using method summary graphs generated by the method described in Section 2.2, and compute points-to information of the graph nodes. When no more edges can be added to the main points-to graph, and no more calling-context is to be expanded, all calling-contexts are expanded and the transitive closure of this graph is computed. Thus, when our algorithm terminates, the main points-to graph contains the final points-to information. The framework of this algorithm is shown in Table 1. Before going to the details of this algorithm, we introduce some concepts first.

3.1 The Representation of Calling-Contexts

A calling-context describes the context under which a method is invoked. In our algorithm, a calling-context is specified by a sequence of call-sites. If the `main()` method calls method m_1 at the call-site cs_1, and m_1 calls the method m_2 at the call-site cs_2, ..., and m_{k-1} calls m_k at the call-site cs_k, the calling-context of this invocation of m_k is $cs_1 : cs_2 : \ldots : cs_k$. The invocation is denoted as $cs_1 : cs_2 : \ldots : cs_k : m_k$. The node corresponding to a local variable v of m_k in this invocation is labeled $cs_1 : cs_2 : \ldots : cs_k : m_k : v$.

3.2 The Nodes and Edges of the Main Points-to Graph

The node set of the main points-to graph includes four types of nodes:

- The nodes corresponding to objects. In our algorithm, objects are identified by their creation statements. All the objects created by the same statement are represented by one node in the main points-to graph.
- The nodes corresponding to reference fields of objects. Our algorithm is field-based. For each object (i.e. object creation statement) o, and each reference field f of o, there is a corresponding node $n_{o.f}$ in the main points-to graph. These nodes are treated as variable nodes, i.e. they hold references to objects.
- The nodes corresponding to static variables. For each static variable in the program under analysis, there is a node in the main points-to graph.
- The nodes corresponding to local variables with calling-context information. A local variable in different calling-contexts corresponds to different nodes.

The last three types of nodes are treated as variable nodes, i.e. they hold references to objects. For each node, our algorithm computes to which object it may point.

The first three types of nodes can be found before the inter-procedure phase begins. So our algorithm use these nodes as the initial node set of the main points-to graph. The last type of nodes are added to the graph dynamically. Our algorithm iteratively expands calling-contexts that are reachable from the `main()` method. Due to the polymorphism of Java, many method invocations may be expanded under one calling-context. Every time a method invocation is expanded, the nodes corresponding to the local variables (decorated with proper calling-context) in this method are added to the main points-to graph.

The types of edges in the main points-to graph are identical to those of summary graphs, i.e. points-to edges, inclusion edges, field-read edges and field-write edges.

3.3 Expanding Calling-Contexts

Our algorithm expands calling-contexts iteratively and adds new nodes to the main points-to graph. A stack named as `worklist` is used to record all the calling-contexts to be expanded. Initially, `worklist` contains only a virtual calling-context, which invokes the function `main()` only. The algorithm gets calling-contexts from `worklist` and then expands them; computes new points-to information; and then puts new calling-contexts to be expanded into `worklist`.

Given a calling-context $cs_0 : \ldots : cs_{k-1} : cs_k$, $cs_0 : \ldots : cs_{k-1}$ is called its *parent* calling-context. Let cs_k be a call-site in the method m, this calling-context is put to `worklist` when the invocation of m under its parent context is expanded. So the nodes $cs_k : rec$, $cs_k : ret$, and node set $\{cs_k : p_i\}$ which represents the real parameters of cs_k already exist in the main points-to graph when $cs_0 : \ldots : cs_{k-1} : cs_k$ is being expanded.

Assuming the calling-context $cs_1 : cs_2 : \ldots : cs_{k-1} : cs_k$ ($cs_{1\ldots k}$ for brief) is being expanded, the algorithm would perform the following steps.

1. The algorithm checks the corresponding part of the parent context $cs_{1\ldots k-1}$ in the main points-to graph and finds out all the objects that the node $cs_{1\ldots k-1} : cs_k$:

rec points to. Based on this information, the algorithm enumerates all the possible methods $g_1, g_2, \ldots, g_n$ that could be invoked at the call-site cs_k under the context $cs_{1\ldots k-1}$. Due to the polymorphism mechanism of Java, a call-cite may invoke different methods according to the class of the receiver. As a result, several methods may be expanded at one call-cite according to the possible classes of the receiver node.

2. For each g_i ($1 \leq i \leq n$), if the invocation $cs_{1\ldots k} : g_i$ is not expanded before, we put the copy of the summary graph of g_i into the main points-to graph.
   - All the variable nodes in the summary graph are cloned into the main graph and renamed. For a node with name n in the summary graph, the new name is $cs_{1\ldots k} : n$.
   - All the edges in the summary graph are also cloned into the main graph.
   - The following inclusion edges are added:
     - from $cs_{1\ldots k} : g_i : this$ to $cs_{1\ldots k-1} : cs_k : rec$,
     - from $cs_{1\ldots k-1} : cs_k : ret$ to $cs_{1\ldots k} : gi : ret$,
     - from $cs_{1\ldots k} : gi : p_j$ to $cs_{1\ldots k-1} : cs_k : a_j$, where p_j represents the formal parameters of g_i, and a_j is real parameters in parent context $cs_{1\ldots k-1}$ for cs_k.
   - For each call-site cs of g_i, a calling-context $cs_{1\ldots k} : cs$ is put into `worklist`.

The algorithm keeps on expanding the calling-contexts till `worklist` is empty. Then the algorithm begins to compute the transitive closure of the main points-to graph.

3.4 Computing the Transitive Closure of the Main Points-To Graph

To compute the transitive closure of the main points-to graph, our algorithm iteratively executes the following two steps till no more edges can be added into the graph.

1. Dealing with field read/write edges in the main points-to graph.
   - For a field-read edge from n to n' labeled f, for each object node n_o such that there is a points-to edge from n' to n_o, we add an inclusion edge from n to $n_{o.f}$, where $n_{o.f}$ is the node corresponding to the field f of the object o.
   - For a field-write edge from n to n' labeled f, for each object node n_o such that there is a points-to edge from n to n_o, we add an inclusion edge from $n_{o.f}$ to n'.
2. Dealing with the inclusion edges in the main points-to graph.
   - For each inclusion edge from n to n', and for each object node n_o such that there is a points-to edge from n' to n_o, a points-to edge from n to n_o is added if there has't been such edge before.
   - If n is like $cnt : cs : rec$, i.e. n is a receiver node of the call-site cs under the calling-context cnt, we should check whether $cnt : cs$ could be expanded again. Denote g as the signature of cs, and C be the class of the object o, the context cnt should be expanded again if $C :: g()$ is resolved to a unexpanded method m under the calling-context $cnt : cs$, we put the calling-context $cnt : cs$ into `worklist` again.

It should be noticed that if some calling-contexts are put into `worklist` during the closure computation, we should go back to the calling-context expanding phase again.

4 Our Solution of Efficiency

Andersen's points-to analysis algorithm has a serious scalability problem due to its time complexity ($O(n^3)$). The CPU time is mainly spent on the iterative process of computing points-to set of each variables.

As mentioned before, our algorithm is a context-sensitive, field-based extension of Andersen's original algorithm. The main points-to graph is constructed dynamically. Our algorithm has to compute the transitive closure dynamically. Thus, it become even more crucial to speed up the closure-computing process. To solve the efficiency problem here, we first make some key properties of the transitive closure computing process clear:

1. Computing the transitive closure in random order will certainly end up with different sums of operations, ranging from linear to $O(n^3)$ [7]. Reducing the number of nodes in the main points-to graph can improve the efficiency of this computation.
2. The most efficient way to compute transitive closure should follow the underlying topological order of nodes induced by the inclusion-edges in the graph.
3. If a sequence of inclusion edges form a cycle, all the nodes in this cycle should have the same points-to object set. They should be collapsed into one node, which greatly accelerate the process of computing transitive closure.

In the rest part of this section, we will describe the optimizations performed on the method summaries and the main points-to graph.

4.1 Cycle Elimination and Node Collapse in the Main Points-To Graph

Reducing the number of nodes by cycle elimination has been the main approach to conquer the efficiency problem of Andersen-style algorithms.

As we mentioned above, all the nodes in a cycle of inclusion edges have same points-to set. They can be collapsed into one node. If we collapse all the cycles in the main points-to graph, the nodes and the inclusion edges form a directed acyclic graph. We can perform a topological sort on this acyclic graph such that if there is an inclusion edge from n to n', n' is prior to n. Computing transitive closure in topological order is much more efficient, because we only need to iterate over the nodes once.

Figure 3 shows the efficacy of the topological sort and cycle-elimination process. The left side is the original points-to graph with 8 nodes. The initial points-to set of each node is shown aside the node. If we compute the points-to set of the nodes following the order $1, 2, 3, 4, 5, 6, 7, 8$, the algorithm gets the final results after 3 iterations, 8 nodes treated in each iteration.

The points-to graph after topological sort and cycle-elimination is shown in the right side. Node 2 and 6 form a cycle, so they are collapsed into one node. The node 2 is selected as their representative with a points-to set $\{o_b\}$. The cycle 3, 5 and 8 are also collapsed and node 3 is selected as the representative, the initial points-to set is $\{o_d\}$.

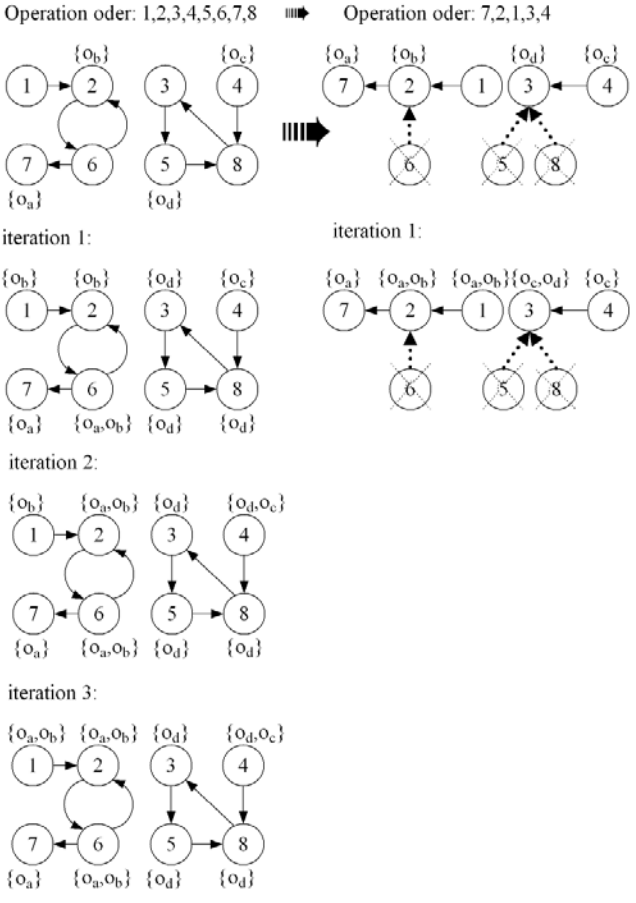


Fig. 3. Node Topo and Cycle Elimination

The nodes are also sorted as $7, 2, 1, 3, 4$. If we treat the nodes following this order, only 1 iteration is needed, and 5 nodes are treated.

We design an algorithm which performs cycle elimination and topological sort simultaneously. This algorithm travels the graph in a depth-first manner following the inclusion edges. When it encounters a node which is already in the current path, a cycle is detected. Then the algorithm collapses the cycle and continues the depth-first exploration. If the algorithm encounters a node of which all the neighbors of inclusion edges are explored (all the neighbors are either collapsed, or added to the sorted list), it adds this node to the end of the sorted list.

In our implementation, when k nodes are collapsed into one representative node, the algorithm marks these nodes as **collapsed** and set a pointer to the representative node, then move all the leaving edges into the representative node. So the time used for collapsing is linear to the nodes be collapsed plus the leaving edges. The time used for depth-first travel is linear to the number of nodes and edges. Suppose that the graph

has n nodes and e edges, and n' nodes are collapsed during cycle elimination. The time complexity of this algorithm is $\mathcal{O}(n + e + n')$. Topological sort and cycle elimination can both improve the efficiency of the transitive-closure computation, and reduce the number of nodes. So in most cases, the benefit of this algorithm weights over its cost.

The algorithm depicted in Figure 2 is an abstract description of the topological sort and cycle-elimination algorithm used in our points-to analysis algorithm. Many technical details, such as merge of edge sets, are omitted for conciseness.

4.2 Optimization in Method Summaries

As described in sub-section 3.3, when a method is being expanded under a calling-context, all the nodes and the inclusion edges in its summary graph are cloned into the main points-to graph. If there is a cycle of inclusion edges, this cycle is also cloned into the main points-to graph. So we can improve our algorithm by eliminate these cycles. We perform two optimizations on summary graphs.

First, we add some extra inclusion edges to summary graphs. In a method, there are many local variables appearing in the left-side of an atomic statement only once. Many of them are temporary variables introduced when program statements are broken down into atomic statements. If u is such a variable that appears only in atomic statement $u = v$, u and v is equivalent, i.e. they should have the same points-to set. We can remove one of them to reduce the size of the summary graph. However, the basic summarization algorithm only add one inclusion edge from u to v, so u and v are treated differently. So for each node n in the summary graph, if n has only one leaving inclusion edge, and the target node is n', and has no field-read edge leaving from n, we add another inclusion edge from n' to n. Thus n and n' forms a cycle of inclusion edges, one of them will be eliminate later.

Second, we perform topological sort and cycle elimination on summary graphs after method summarization. In a method summary graph, we use four kinds of edges to describe the relations between the nodes. During the calling-context expanding phase, method summary graphs are cloned into the main points-to graph. The inclusion edges in summary graphs are also cloned into the main graph. So the topological orders between summary graph nodes induced by the inclusion edges in summary graphs are reserved in the main points-to graph. Furthermore, cycles of inclusion edges in a method summary graph will still be cycles in the main graph. Thus, after we add some extra inclusion edges into summary graphs as described above, we perform a topological sort and cycle-elimination in each summary graph using the same algorithm depicted in Figure 2.

As size of method summary graphs is much smaller than the main points-to graph, overhead of optimizations shrinks tremendously. A summary graph may be cloned many times into the main graph, so the benefit of these optimizations is two folds:

- It reduces the size of summary graphs, thus reduces the size of the main graph.
- The circles in summary graphs are eliminated once for all with a smaller cost.

Table 2. Node Topo and Cycle Elimination

INPUT: The main points-to graph G;
OUTPUT: The main points-to graph G with all cycles are eliminated;
A topological sort of the nodes in G, stored in $sortlist$;

TOPOSORTANDCIRCLEELIMINATION (**G**)

for each node $n \in G$ **do**
mark n as unvisited
$sortedlist := \langle\rangle$
while (there are some unvisited nodes in G) **do**
begin
$curnode$ = a unvisited node in G
$trace := \langle curnode \rangle$
while ($trace$ is not empty) **do**
begin
if ($curnode$ has unvisited successors)
begin
n := a unvisited successor of $curnode$;
if (n = the ith element of $trace$)
begin
CIRCLECOLLAPSE(i);
$trace$:= the prefix of $trace$ with length i;
end else begin
append n to the end of $trace$;
$curnode := n$;
mark n as visited;
end
end else begin
append $curnode$ to the end of $sortedlist$;
remove the last element of $trace$;
$curnode$:= last node of $trace$;
end
end /*of *while (trace is not empty)*/
end /* of *while (there are some unvisited nodes in G)*/

CIRCLECOLLAPSE(i)
begin
n := the ith element of $trace$;
for each node n' from the ith element of $trace$ to the end **do**
begin
Set n as the representative of n';
Combine the leaving edges of n' to the leaving edge set of n;
end
end

5 Experimental Result and Evaluation

The inclusion-based points-to analysis algorithm presented in this paper has been implemented as a plug-in of the Eclipse IDE. We use Java Design Tool (JDT) as the source code parser, which transforms all the source code under analysis into abstract syntax trees (ASTs). These ASTs contain the information about the structure of the programs and types of variables. Algorithm then traverse these ASTs to get the method summaries. The main points-to graph is generated based on these method summaries.

The experiments are performed on a PC with a 2.2GHz Intel duo core CPU and 3.2 GB of memory, running Windows XP and Java VM build 1.5.0. We run all the cases with a heap size of 512MB.

Table 3 shows the performance data of our analysis on a variety of benchmarks. These nine benchmarks are from the standard SPECjvm [Standard Performance Evaluation Corporation]2008 benchmark suite.

Table 3. Analysis result on the benchmarks

	size			time(msec)					
Bechmark	cls	meth	LOC	summary	opt	no_opt	pre_meth	cons	cyc
compress	77	826	10479	3500	31	47	28	29	8
crypto.aes	66	760	9867	3172	94	156	24	154	16
crypto.signverify	66	760	9841	2922	93	125	21	116	12
MPEGaudio	66	757	9708	2891	31	47	25	26	10
scimark.fft	71	803	10581	3219	63	78	46	61	22
$scimark_{MonteCarlo}$	70	794	10425	3031	32	47	36	38	9
scimark.lu	70	804	10651	3063	46	63	48	51	46
scimark.sor	71	796	10428	3156	47	78	41	44	21
sparse	70	796	10473	3125	47	63	41	45	18
Validation	67	770	9910	3047	47	62	36	51	13

The first three columns of the table are the numbers of classes, methods, and LOC of the benchmarks. Our algorithm analyzes only those methods reachable from the root set, while those invocations related to Java standard libraries are simply ignored.

The next three columns give the run time of our algorithm: summarization time, main algorithm time with and without the optimizations described in Section 4. Summarization time is about 3 seconds. The analysis algorithm speeds up significantly, by a factor between 1.3 and 1.7, depending on the structure of program. The last three columns give a general status of our results. These are the facts greatly affecting the result of our approach, which is amount of methods that need to be pre-treated, amount of contexts that are generated, and amount of cycles that have been eliminated during our analysis, respectively. Generally speaking, cycle elimination in summary graph is time consuming. But these operations only needs to be done once per method(if exists). The ratio of contexts and pre-treat methods is the key factor to buffer this consuming.

Our algorithm spend about 3 seconds to get the summaries of methods, and analyzes within 0.1 seconds over 10K LOC. It's faster than other presented algorithms, such

as [6]. Reporting time of this algorithms ranges from 5 seconds to a dozen seconds. Though their experiments are performed on an early version of SPECjvm, which is SPECjvm1.03, results are comparable due to the same scale of code size.

The result here shows that our algorithm can analyze Java code with a reasonable memory requirement and a relative low time expense.

6 Related Work

The basic idea in our algorithm is mainly assemble with the one proposed by Heintze and Tardieu [5]. They introduce a new algorithm for computing dynamic transitive closures. As new inclusion edges are added to the constraint graph from the indirect constraints, their corresponding new transitive edges are not added to the graph at the same time. Instead, the constraint graph retains its pre-transitive form, and during the analysis, indirect constraints are resolved via reachability queries on the graph. A lot of queries will result in the same transitive edge, which is the main drawback of their work. They won't know whether the result of a query is worthy until the end of the query. The experiment result showed in [5] In our algorithm, we summarize each method and avoid the redundant operations. This method speedups our algorithm remarkably.

Faehndrich et al. [9] introduce an online cycle detection algorithm which play the cycle detection at every edge insertion. Pearce propose an more efficient analysis [7]. In order to avoid cycle detection at every edge insertion, the algorithm dynamically maintains a topological ordering of the constraint graph. Only a newly-inserted edge that violates the current ordering could possibly create a cycle, so only in this case are cycle detection and topological re-ordering performed. Later, they improve the algorithm [8], by introducing a periodically sweep, which sweep the entire constraint graph to detect and collapse any cycles that have formed since the last sweep.

Furthermore, we combine the cycle elimination and node topology together. So besides reduction of node numbers, our method can also reduce the number of iterations needed for transitive-closure computation.

All the related work above are done for context-insensitive points-to analysis, of whom, the constraint graph has a fixed size. Situation for context-sensitive points-to analysis is more complex. Our solution handles this challenge successfully.

We use the pithiness version of points-to set for each method by pre-treating the method summaries. Many cycles has been eliminated during the pre-treatment. It makes the final main graph construction with a much smaller input size.

7 Conclusion

We have presented an efficient inclusion-based points-to analysis for Java. Our algorithm handles Java language features such as inheritance, object fields, and aggregate objects. Our algorithm uses a method summary to briefly describe the constrains of each method, and pretreat each summary so as to eliminate the cycles in method scope. This paper presents empirical results which demonstrate that our algorithm is scalable to large code bases without sacrificing much accuracy.

References

1. Andersen, L.O.: Program Analysis and Specialization for the C Programming Language. PhD thesis, University of Copenhagen, DIKU (1994)
2. Steensgaards, B.: Points-to analysis in almost linear time. In: Proceedings of the ACM Symposium on Principles of Programming Languages (POPL), pp. 32–41. ACM, New York (1996)
3. Ryder, B.G.: Dimensions of Precision in Reference Analysis of Object-Oriented Programming Languages. In: Hedin, G. (ed.) CC 2003. LNCS, vol. 2622, pp. 126–137. Springer, Heidelberg (2003)
4. Sridharan, M., Gopan, D., Shan, L., Bodik, R.: Demand-driven points-to analysis for Java. In: Proceedings of the 20th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (October 2005)
5. Heintze, N., Tardieu, O.: Ultra-fast aliasing analysis using CLA: A million lines of C code. In: Proceedings of the Conference on Programming Language Design and Implementation (PLDI), pp. 146–161 (2001)
6. Whaley, J., Lam, M.: An efficient inclusion-based points-to analysis for strictly-typed languages. In: Hermenegildo, M.V., Puebla, G. (eds.) SAS 2002. LNCS, vol. 2477, pp. 180–195. Springer, Heidelberg (2002)
7. Pearce, D.J., Kelly, P.H.J., Hankin, C.: Online cycle detection and difference propagation for pointer analysis. In: 3rd International IEEE Workshop on Source Code Analysis and Manipulation, SCAM (2003)
8. Pearce, D.J., Kelly, P.H.J., Hankin, C.: Efficient Field- Sensitive Pointer Analysis for C. In: ACM Workshop on Program Analysis for Software Tools and Engineering, PASTE (2004)
9. Faehndrich, M., Foster, J.S., Su, Z., Aiken, A.: Partial online cycle elimination in inclusion constraint graphs. In: PLDI 1098: Proceedings of the ACM SIGPLAN 1998 conference on Programming language design and implementation, pp. 85–96. ACM Press, New York (1998)
10. Ramalingam, G.: The Undecidability of Aliasing. ACM Trans. Program. Lang. Syst. 16(5), 1467–1471 (1994)
11. Landi, W.: Undecidability of static analysis. ACM Letters on Programming Languages and Systems 1(4), 323–337 (1992)

A Formal Analysis of the Web Services Atomic Transaction Protocol with UPPAAL

Anders P. Ravn, Jiří Srba*, and Saleem Vighio**

Department of Computer Science, Aalborg University,
Selma Lagerlöfs Vej 300, DK-9220 Aalborg East, Denmark
{apr,srba,vighio}@cs.aau.dk

Abstract. We present a formal analysis of the Web Services Atomic Transaction (WS-AT) protocol. WS-AT is a part of the WS-Coordination framework and describes an algorithm for reaching agreement on the outcome of a distributed transaction. The protocol is modelled and verified using the model checker UPPAAL. Our model is based on an already available formalization using the mathematical language TLA$^+$ where the protocol was verified using the model checker TLC. We discuss the key aspects of these two approaches, including the characteristics of the specification languages, the performances of the tools, and the robustness of the specifications with respect to extensions.

1 Introduction

Web Services (WS) are distributed applications that interoperate across heterogeneous networks and provide services that are hosted and executed on remote systems. Web services infrastructures employ one or more layers of a web service protocol stack (see e.g. [8]), containing various standardization initiatives on aspects which need to be implemented and described in web services environments. Many protocols in the stack use the SOAP [15] conventions and are currently at various adoption stages, ranging from approved standards to proposals.

Several protocols for web services require transactional support in order to preserve consistency. A classical transaction terminates with two possible outcomes: *committed* or *aborted*. In the committed case, the outcome is made persistent and visible outside the transaction, whereas in the aborted case, all the actions taken during the transaction are cancelled. Standards for supporting transactions among web services include the WS-Coordination framework [9] developed by BEA Systems, IBM and Microsoft. Web Services Atomic Transaction (WS-AT) is a part of this framework. This specification defines three coordination protocols that are used by distributed applications which require consistent agreements on the outcome of short-lived distributed activities.

* The author is partially supported by the Ministry of Education of The Czech Republic, project 1M0545 — Institute for Theoretical Computer Science.

** The author is supported by Quaid-e-Awam University of Engineering, Science, and Technology, Nawabshah, Pakistan.

T. Margaria and B. Steffen (Eds.): ISoLA 2010, Part I, LNCS 6415, pp. 579–593, 2010.

Web services protocols are in general nontrivial and their correctness is not obvious. Therefore we model WS-AT as a network of abstract state machines communicating via shared variables and, beside some other properties, verify its correctness using the model checker Uppaal [1]. Verification of communication protocols is in general not a new topic (see e.g. [3]) but WS-AT was formally specified only recently, and analysed in [4] using the language TLA$^+$ [5] and its model checker TLC [7]. The TLA$^+$ formalization of the protocol remained very useful for the creation of our Uppaal model. In fact, we have transferred the state transition tables specified in TLA$^+$ into our Uppaal model so that we can make a fair comparison of the two specification languages.

We compare the TLA$^+$ model with our abstract state machine model with respect to several criteria. First of all, we consider the performance of the verification tools TLC and Uppaal. We were able to verify the protocol for up to five participants but the verification in Uppaal was significantly faster. Then we discuss the foundations of the two approaches as TLA$^+$ is based on a formal mathematical language while Uppaal automata rely on imperative programming constructs and transition graphs. We mention the expressiveness of these formalisms and consider the robustness of the models with respect to a wider applicability in other protocols with a particular focus on measuring the quality of service. In conclusion, the two formalization languages complement each other and we discuss a combination of these approaches for future applications in the specification and analysis of web services protocols.

The rest of the paper is organized as follows. In Section 2, we give an overview of the web services atomic transaction protocol. Section 3 discusses the TLA$^+$ modelling approach. A Uppaal model of the protocol is presented in Section 4. Model properties and verification results are discussed in Section 5. Finally, Section 6 provides a detailed comparison and discusses ideas for future work.

2 Overview of WS-Atomic Transaction Protocol

WS-Coordination [9] is a specification framework for the description of protocols that coordinate the actions of applications in a distributed environment. WS-Atomic Transaction [10] (or WS-AT for short) is a part of this framework that defines an atomic transaction coordination type based on the well-known ACID (Atomicity, Consistency, Isolation, Durability) principle [2]. WS-AT defines three specific agreement coordination protocols: Completion, Volatile two-phase commit, and Durable two-phase commit. The goal of the protocols is to reach an agreement between a protocol initiator and a number of protocol participants on whether the transaction should be committed or aborted. It is following the "all or nothing" policy with no compensation mechanism. All communication between the initiator and the participants is established via a transaction coordinator. See Figure 1 for the parties involved in the protocol and their communication.

The *completion protocol* is a simple communication scheme between the initiator and the coordinator. It is essentially used by the initiator to ask the

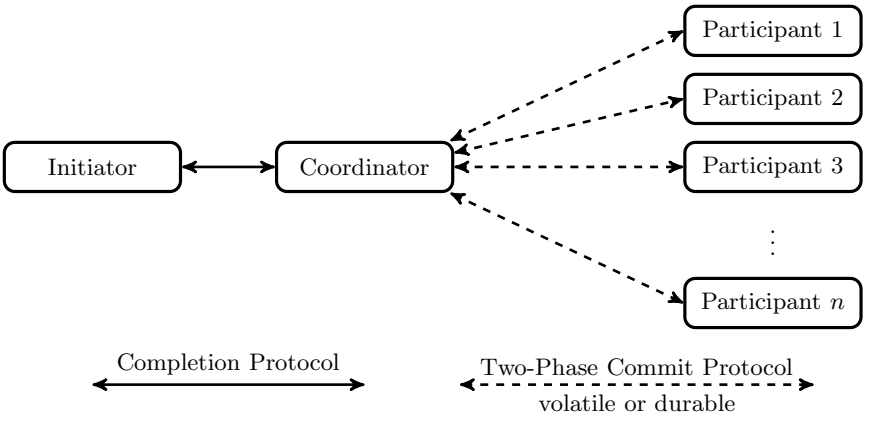


Fig. 1. WS-AT communication scheme

coordinator to try to commit or abort the transaction. The other two protocols are both based on the *two-phase commit* (2PC) protocol (see Figure 2) which coordinates registered participants in reaching their commit or abort decisions. First, coordinator invites the registered participants to prepare for committing the transaction, on which a participant can either vote for abort and terminate, or answer that it is either prepared to commit or is read-only (meaning that the participant's commit does not require any further action). The second phase of the protocol handles the actual commit, provided that the first phase was successful. In the *volatile* variant of 2PC protocol, the specification describes the communication between the coordinator and participants managing volatile resources (like a cache register). The *durable* variant deals with the coordinator-participant conversation for participants managing durable resources (like a database register). The WS-AT protocol combines the two protocols into a new three-phase (i) prepare volatile, (ii) prepare durable and (iii) commit protocol. Moreover it allows the participants to register for the protocol at any time before the prepare phase of their respective category is completed; thus as an example, durable participants can still register while the registration of volatile participants is already closed.

3 Formalization and Modelling of the Protocol

The WS-AT standard [10] provides a high-level description of the protocol. It is a collection of protocol behaviours described in English accompanied by diagrams like the two-phase commit state transition graph presented in Figure 2 and state tables for the parties involved in the protocol, see part a) in Figure 3. Unfortunately, there is no generally accepted method for formally specifying WS protocols and, as documented in [4], the WS-AT description is not sufficiently precise for a direct formalization. For example, the roles in the protocol are not sufficiently separated from each other, which causes confusion in the protocol

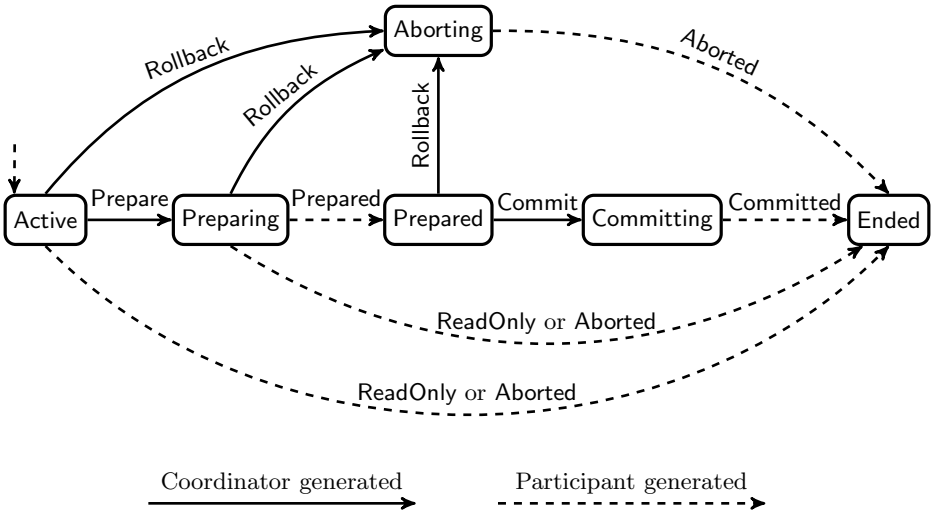


Fig. 2. Two-phase commit state transition graph

description as well as in the state tables. The description is silent also about what kind of communication between the parties in the protocol is assumed, as well as what data the coordinator stores about each participant.

Let us take a look at Figure 3 a) describing how the transaction coordinator handles the message Prepared arriving from some participant. The WS-AT description says that if the coordinator is in the state Preparing and receives the message Prepared, then it should register the vote and change its state to Prepared. How this rule should be interpreted when another participant sends its Prepared message is not explicitly formulated and the WS-AT description says only that a coordinator with multiple participants can be understood as a collection of independent coordinator state machines, each with its own state. Furthermore, the state tables do not describe the details of how and when the decision about commit or abort is made.

The WS-AT description can be formalized using the TLA$^+$ language as shown in Figure 3 b) taken from [4]. TLA$^+$ [5] is a formal mathematical language for specifying high-level descriptions of distributed systems. The language is very expressive; it uses predicate logic with first order quantification, which allows for expressing the protocol behaviour in a rather elegant way. There are no built-in constructs for protocol primitives like message passing, but they can be encoded using the mathematical formalism, for example as sets in case of the message passing. The TLA$^+$ expression in Figure 3 b) describes that if there is a message m of the type Prepared in the set *msgs* containing all messages sent so far, and the transaction coordinator is in the state preparingVolatile and the sender of the message is registered as volatile, or the coordinator is in preparingDurable and the sender is registered as durable, then the coordinator will note that this particular participant is now prepared to commit the transaction. In TLA$^+$ it is necessary

a) WS-AT:
The coordinator accepts the message Prepared. Upon receipt of this notification, the coordinator knows the participant is prepared and votes to commit the transaction.

Inbound Events	**States**			
	...	Preparing	Prepared	...
⋮	⋮	⋮	⋮	⋮
Prepared	...	Record vote; goto Prepared	...	...
⋮	⋮	⋮	⋮	⋮

b) TLA$^+$:

$\exists m \in msgs : m.type =$ "Prepared"
  $\wedge$ CASE
    $\vee \wedge$ $tcData.st =$ "preparingVolatile"
      $\wedge$ $tcData.reg[m.src] =$ "volatile"
    $\vee \wedge$ $tcData.st =$ "preparingDurable"
      $\wedge$ $tcData.reg[m.src] =$ "durable"
  $\longrightarrow$
    $\wedge$ $tcData' =$ [tcData EXCEPT !.reg[m.src] = "prepared"]
    $\wedge$ UNCHANGED $msgs$
  $\Box$
  ⋮

c) Uppaal Timed Automata:
An edge in the coordinator automaton with the construct select `parId: Participant`, guard `guard9(parId)` and update `action9(parId)`, where

```
bool guard9(Participant parId) {
 if ((msgSrc[parId][PREPARED] == true) &&
  ((tcData.st == TC_PREPARING_VOLATILE && tcData.reg[parId] == TC_VOLATILE)
     ||
  (tcData.st == TC_PREPARING_DURABLE && tcData.reg[parId] == TC_DURABLE)))
   return true;
 return false;
}

void action9(Participant parId) {
 tcData.reg[parId] = TC_PREPARED;
}
```

Fig. 3. Specification of a selected WS-AT transition in TLA$^+$ and UPPAAL

to explicitly assert that this rule does not change the current set of messages. This is given by the clause "UNCHANGED *msgs*".

During the formalization of WS-AT in TLA$^+$, the authors in [4] had to make a few design decisions. First of all, it was agreed that the completion protocol between the initiator and the coordinator is modelled via internal communication as one single process. The 2PC protocol is modelled via unreliable asynchronous

message passing, where the messages can be reordered, lost or duplicated. Internal timing events like "expires times out" in the state tables are modelled via nondeterminism, which provides a safe over-approximation of the behaviour but disallows the verification of any time-dependent properties. The full TLA$^+$ model is described in [4] and its correctness has been verified using the TLC model checker [7] for up to four protocol participants. In [4, 7], it is concluded that the formalization of the protocol was nontrivial and a discussion with designers involved in the formulation of WS-AT was necessary, because the WS-AT definition employs informal descriptions being imprecise, ambiguous and often fail to consider unusual cases.

The authors in [4] support their choice of TLA$^+$ as modelling language by the argument that there is a place in the specification where one process depends on the internal state of another process, and that this can be hard to model in some languages designed expressly for distributed systems.

In the section to follow, we explain an alternative way to model the WS-AT protocol with a network of state machines as provided by the model checker Uppaal [12, 1] (see Figure 3 part c) for an example of Uppaal syntax) and compare the advantages and disadvantages of both approaches. We also explain how the difficulty with rules that depend on internal states of other processes can be solved in our approach via the use of global variables and a special form of process templates.

4 The UPPAAL Model

In this section we provide the details about our Uppaal model of WS-AT protocol. Uppaal [12] is a tool for modelling, simulation and verification of networks of timed automata. The language allows to describe communicating abstract state machines with handshake synchronization and communication via shared variables. It provides a powerful C-like syntax for describing guards and updates on transitions. Uppaal allows also real time clocks. However, this feature is not used in the present model. We refer the reader to [1] for a thorough introduction to the Uppaal modelling language.

The protocol model in Uppaal consists of a process that models the initiator together with the transaction coordinator (TC for short) and a participant template that can be instantiated to as many participant processes as we want to consider.

4.1 Global Declarations

Global variable declarations of the protocol model contain the set of states for the initiator, for TC, and for the participants. We also define a set of registration types and outcomes for TC and the participants. Finally, we model the set of messages sent between TC and participants as a bit-vector.

The protocol model consists of n participants. A type `Participant` identifies a participant using the indices among $0, 1, \ldots, n-1$.

```
const int NO_OF_PARTICIPANTS = n;
typedef int[0,NO_OF_PARTICIPANTS-1] Participant;
```

Initiator and TC related declarations. The initiator's state is stored in the variable `iState` which may contain one of the following values.

$$\texttt{iState} \in \{\texttt{I\_ACTIVE, I\_COMMITTED, I\_ABORTED, I\_COMPLETING}\}$$

The information about TC is stored in the variable `tcData`. It is defined as a variable of the record type `DataTC`.

```
typedef struct {StateTC st; RegTC reg[Participant];
                ResTC res;} DataTC;
DataTC tcData;
```

The record type `DataTC` contains three components.

- A variable `st` of type `StateTC` represents all possible control states of TC.

$$\texttt{st} \in \{\texttt{TC\_ACTIVE, TC\_PREPARING\_VOLATILE, TC\_PREPARING\_DURABLE, TC\_ABORTING, TC\_COMMITTING, TC\_ENDED}\}$$

- The array `reg[Participant]` is defined as `RegTC` type and stores the registration state (known to the TC) for each participant `parId`.

$$\texttt{reg[parId]} \in \{\texttt{TC\_UNREGISTERED, TC\_VOLATILE, TC\_DURABLE, TC\_PREPARED, TC\_READ\_ONLY, TC\_COMMITTED}\}$$

- Finally, a variable `res` of type `ResTC` represents the outcome of the protocol.

$$\texttt{res} \in \{\texttt{TC\_COMMITTED\_RES, TC\_ABORTED\_RES }\}$$

Participants' related declarations. The array `pData[Participant]` represents the data maintained by each participant and is declared as `DataP` record type.

```
typedef struct {StateP st; RegP reg; ResP res;} DataP;
DataP pData[Participant];
```

The record type `DataP` contains three components.

- A variable `st` of type `StateP` represents the control states of a participant.

$$\texttt{st} \in \{\texttt{P\_UNREGISTERED, P\_PREPARED, P\_REGISTERING, P\_ACTIVE, P\_PREPARING, P\_ENDED}\}$$

- A variable `reg` of type `RegP` records the registration status of a participant.

$$\texttt{reg} \in \{\texttt{P\_VOLATILE, P\_DURABLE}\}$$

- Finally, `res` of type `ResP` represents the outcome of the protocol as recorded by the given participant.

$$\texttt{res} \in \{\texttt{P\_READ\_ONLY, P\_COMMITTED, P\_ABORTED }\}$$

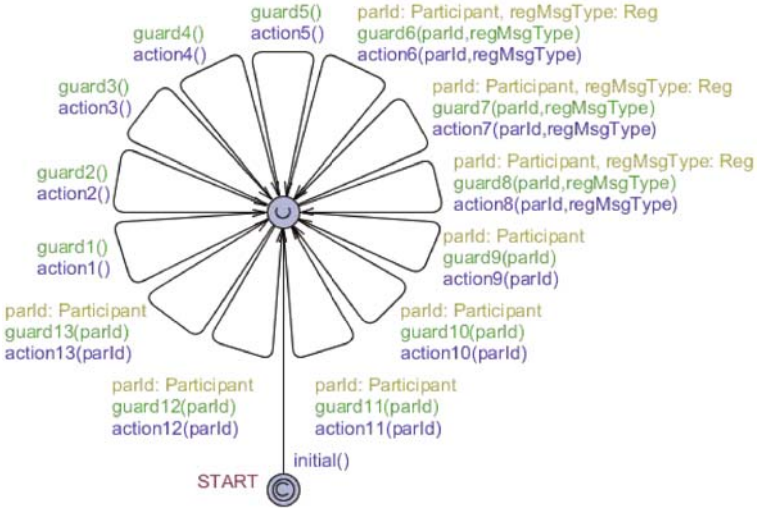


Fig. 4. Initiator-Coordinator process

4.2 Messages

Following the TLA$^+$ model, we decided to model the completion protocol between the initiator and the TC by a direct communication. For the communication between TC and participants, we adopted the model of asynchronous message passing where messages can be reordered, duplicated or lost. Unlike in TLA$^+$, we model messages as two dimensional Boolean arrays. There are two types of messages, either sent by the TC to a concrete participant or sent by a participant to the TC.

- The array `msgDest[Participant][MsgsTC]` stores messages of type `MsgsTC` sent from the TC to a participant where `MsgsTC` = {`REGISTER_RESPONSE, PREPARE, COMMIT, ROLLBACK`}. Given a participant `parId` and a message `msg` of type `MsgsTC`, the array element `msgDest[parId][msg]` has the value true if and only if the TC has already sent the message `msg` to the participant `parId`.
- The array `msgSrc[Participant][MsgsP]` represents messages of type `MsgsP` sent from a participant to TC where `MsgsP` = {`PREPARED, READ_ONLY, COMMITTED, ABORTED, REGISTER_VOLATILE, REGISTER_DURABLE`}. As before, if a participant `parId` has already sent a message `msg` to the TC, then `msgSrc[parId][msg]` has the value true, otherwise it is false.

This representation ensures that duplicate messages are ignored, and that the arrival order of messages is ignored as well.

4.3 Initiator-Coordinator Process

The model for the Initiator-Coordinator process is shown in Figure 4. The execution starts in the location START from which the protocol is set to its initial values

by the function `initial()`. After this initial phase, the model has just one location which is urgent (no time elapse is allowed). Each of the transitions in the model has a guard and an update, both modelled as a function in C-like code. Due to the space limitation we present here in detail only rule 9, as already displayed in Figure 3 c). The rule has a select statement `parId: Participant` which is a convenient UPPAAL abbreviation for a set of transitions where the `parId` variable is instantiated to all possible participant identities (as defined by the type `Participant`).

Referring to Figure 3 c), the boolean function `guard9(parId)` has a parameter `parId`. The function checks if the TC has already received a prepared message from the selected participant by the test `(msgSrc[parId][PREPARED] == true)`. It also checks if the TC is currently in the preparing volatile phase of the protocol `(tcData.st == TC_PREPARING_VOLATILE)` and the registration state for the participant that the TC has recorded is volatile `(tcData.reg[parId] == TC_VOLATILE)`. The guard is satisfied also if the current TC's state is preparing durable `(tcData.st == TC_PREPARING_DURABLE)` and TC's recorded registration state of the participant is durable `(tcData.reg[parId] == TC_DURABLE)`. If one of these conditions is satisfied then the guard returns true, otherwise it returns false.

If the guard `guard9(parId)` is true then the transition can be executed and the function `action9(parId)` is called for the selected participant. The call of `action9(parId)` simply sets the TC's registration state for the participant `parId` to prepared `(tcData.reg[parId] = TC_PREPARED)`.

The reader may observe that some rules in Figure 4 do not have any select statements, others select a `parId`, and a few select the registration type of messages `regMsgType` as well. The full UPPAAL model is available as [11].

4.4 Participant Process

The template for the participants has a similar shape as the initiator-coordinator template. The final UPPAAL model contains one copy of this template for each participant in the network. Like the initiator-coordinator process, the participant process also starts in the location `START` and performs the initialization first. The participant model then consists of loop-transitions with guards and updates (actions) numbered from 14 to 22. Consult [11] for the complete model.

Following the TLA$^+$ specification, we encoded into the rules the behaviour of each participant. An example of such a rule is a situation when the participant identified as `id` receives a rollback message from the TC. As long as the participant is in one of the four prescribed states (defined in the code to follow), it will be able to read this message and end the transaction with the aborted outcome and confirm this by sending a message to the TC.

```
bool guard22() {
  return (msgDest[id][ROLLBACK] == true &&
     (pData[id].st == P_REGISTERING || pData[id].st == P_ACTIVE ||
      pData[id].st == P_PREPARING || pData[id].st == P_PREPARED))
}
```

```
void action22() {
  pData[id].st = P_ENDED; pData[id].res = P_ABORTED;
  msgSrc[id][ABORTED] = true;
}
```

The other participant rules follow a similar pattern.

5 Model Properties and Verification Results

We discuss now the properties of the UPPAAL WS-AT model we described in the previous section and compare its verification results with those of the TLC model checker.

5.1 Model Properties

Consistency: The main correctness requirement is that the participants together with the initiator unanimously agree to commit or abort the transaction. This property is called consistency in [4], and its formulation in TLA$^+$ can be

$$
\begin{array}{l}
Consistency \triangleq \\
\land\ (iState = \text{“committed”}) \\
\quad \Rightarrow\ \lor\ \ \land\ tcData.st = \text{“ended”} \\
\qquad\qquad \land\ tcData.res = \text{“committed”} \\
\qquad\qquad \land\ \forall\, p\ \in\ Participant : \\
\qquad\qquad\qquad \lor\ pData[p].st = \text{“unregistered”} \\
\qquad\qquad\qquad \lor\ \ \land\ pData[p].st = \text{“ended”} \\
\qquad\qquad\qquad\qquad \land\ pData[p].res = \{\text{“?”}\ ,\ \text{“committed”}\} \\
\qquad\ \lor\ \ \land\ tcData.st = \text{“committing”} \\
\qquad\qquad \land\ \forall\, p\ \in\ Participant : \\
\qquad\qquad\qquad \lor\ pData[p].st = \{\text{“unregistered”}\ ,\ \text{“prepared”}\} \\
\qquad\qquad\qquad \lor\ \ \land\ pData[p].st = \text{“ended”} \\
\qquad\qquad\qquad\qquad \land\ pData[p].res = \{\text{“?”}\ ,\ \text{“committed”}\} \\
\land\ \forall\, p\ \in\ Participant : \\
\quad \land\ pData[p].st = \text{“ended”} \\
\quad \land\ pData[p].res = \text{“committed”} \\
\quad \Rightarrow\ \ \land\ iState = \text{“committed”} \\
\qquad \land\ \lor\ \land\ tcData.st = \text{“ended”} \\
\qquad\qquad\ \land\ tcData.res = \text{“committed”} \\
\qquad\qquad\ \land\ iState = \text{“committed”} \\
\qquad\quad \lor\ tcData.st = \text{“committing”} \\
\qquad \land\ \forall\, pp\ \in\ Participant : \\
\qquad\qquad \lor\ pData[pp].st = \{\text{“unregistered”}\ ,\ \text{“prepared”}\} \\
\qquad\qquad \lor\ \ \land\ pData[pp].st = \text{“ended”} \\
\qquad\qquad\qquad \land\ pData[pp].res = \{\text{“?”}\ ,\ \text{“committed”}\}
\end{array}
$$

Fig. 5. Consistency Property in TLA$^+$

```
bool Consistency() {
 return InitiatorCommittedOK() && ParticipantCommittedOK();
}
bool InitiatorCommittedOK() {
 return iState != I_COMMITTED ||
      ( tcData.st == TC_ENDED && tcData.res == TC_COMMITTED_RES &&
             AllParticipantsCommitted() ) ||
      ( tcData.st == TC_COMMITTING && AllParticipantsCommitting() );
}
bool AllParticipantsCommitted() {
 for (p=0; p<NO_OF_PARTICIPANTS; p++)
  if (!(pData[p].st == P_UNREGISTERED || (pData[p].st == P_ENDED &&
        (pData[p].res == P_READ_ONLY || pData[p].res == P_COMMITTED))))
  return false;
return true;
}
bool AllParticipantsCommitting() {
 for (p=0; p<NO_OF_PARTICIPANTS; p++)
  if (!((pData[p].st == P_UNREGISTERED || pData[p].st == P_PREPARED) ||
      (pData[p].st == P_ENDED &&
       (pData[p].res == P_READ_ONLY || pData[p].res == P_COMMITTED))))
  return false;
return true;
}
bool ParticipantCommittedOK() {
 for (p=0; p<NO_OF_PARTICIPANTS; p++)
  if (pData[p].st == P_ENDED && pData[p].res == P_COMMITTED) {
    if ( !InitCoorCommittedOrCommiting() || !AllParticipantsCommitting() )
      return false;
 }
return true;
}
bool InitCoorCommittedOrCommiting() {
 return iState == I_COMMITTED &&
        ( (tcData.st == TC_ENDED && tcData.res == TC_COMMITTED_RES) ||
          tcData.st == TC_COMMITTING);
}
```

Fig. 6. Consistency property in UPPAAL

seen in Figure 5. Consistency is a safety property, and it is expressed by an invariant assertion. It states that the protocol is never in an inconsistent configuration where one process thinks that the transaction is committed while another process claims that it was aborted. There are two separate conjuncts in the invariant, one asserting what should be true if the initiator reached the decision to commit, and the other one asserting what is true if a participant has reached the commit decision.

We verified the same consistency property given in [4] by reformulating it to the UPPAAL syntax. The UPPAAL expression for checking consistency is given as the function `Consistency()` in Figure 6. The query is then formulated in UPPAAL's CTL logic as `A□ Consistency()` which checks whether on all computations every state satisfies the consistency invariant. It is no surprise that the WS-AT protocol satisfies this property as it is a rather standard protocol and it was recently verified using the model checker TLC [4], resulting in some modifications and improvements in the official specification.

Rules Usage: The next question one can ask is whether all rules implemented in the UPPAAL model are actually necessary, in other words if for any given rule there is some execution where the rule is actually used.

For this purpose we introduce an `observer`, which is a function added to the update of every transition in our model which simply records the number of the rule that was executed.

```
typedef int[1,22] Rules; bool flag[Rules];
void observer(int x) {   flag[x] = true; }
```

As we numbered all the rules in our models (see e.g. Figure 4) it is now easy to add the calls `observer(1)`, ..., `observer(22)` to the updates of the transitions representing the rules 1 to 22, respectively. Now in order to verify whether for example the rule 9 is ever used, we ask UPPAAL the query `E◇ flag[9]`. In this way we verified (again as expected) that all rules specified in the protocol are actually used at some execution.

5.2 Performance Results

We measured the time needed for the verification of the consistency property, which is the most time demanding one as it searches the whole state-space. The tests were performed on a iMac 27in, 4 GB 1067 MHZ RAM, 3.06 GHz Intel Core 2 Duo and Leopard Snow operating system. We used UPPAAL 4.1.2 and TLA Toolbox version 1.1.0, both with the default settings. The results are shown below along with the results obtained using the TLC model checker for the protocol description given in [4]. Execution times are rounded up to seconds and we also report on the number of explored states.

Performance Results for Checking Consistency				
Number of participants	**TLC**		**UPPAAL**	
	Time	States	Time	States
1	1s	132	1s	143
2	1s	2 082	1s	2 621
3	6s	32 244	2s	50 537
4	1m 49s	504 306	33s	1 014 497
5	40m 37s	8 000 412	14m 36s	21 100 793

Tool	**TLC**	**UPPAAL**
specification language	TLA$^+$	timed automata network with shared variables
necessary user's background	mathematical	programming
expressiveness of spec. language	very expressive, infinite sets, relations, quantifiers, co-inductive approach	restricted, communicating state machines, C-like (but finite) data-structures, inductive approach
model checker characteristics	restricted to bounded domains, exhaustive search	verifies the full specification language (with time)
modelling/verification speed	fast modelling, slower verification	slower modelling, faster verification
verification of time/cost features	manual encoding necessary but no verification support	straightforward modelling and state-of-the-art verification support
parameterized reasoning	modelling yes, verification no	modelling yes, verification no

Fig. 7. Comparison of the model checkers TLC and UPPAAL

Comparison of the verification results indicates that UPPAAL is more efficient than the model checker TLC in terms of execution time, even though it actually explores more states. Beyond five participants, it is almost certain that UPPAAL will run out of RAM (and start swapping) and TLC may take a very long time (probably days). However, we have not tried to optimize the UPPAAL model in any way yet and we believe that related tools, like for example UPPAAL CoVer [14], may significantly improve its performance.

6 Comparison and Conclusion

We conclude by discussing the key aspects of the two approaches presented for formalization and verification of WS protocols. A summary table is in Figure 7.

Perhaps the main difference is that TLC can analyse (a subset of) the TLA$^+$ language which is based on mathematical reasoning: first order logic and a simple set theory. Reading of TLA$^+$ specifications requires training, but the authors claim that it is about as difficult as learning a new programming language [4]. The UPPAAL model of WS-AT may look more familiar to engineers even without any prior training in concurrency theory. The model in fact uses only a limited set of UPPAAL primitives. For example no synchronization between processes is employed as all message passing is asynchronous and modelled using shared variables. The rules of the protocol are encoded in a C-like programming language.

Yet, Uppaal requires a lower-level encoding of some protocol fragments like message passing. Messages in Uppaal are encoded as bit-vectors, while TLA$^+$ offers an elegant and easy to read set notation. This necessarily implies a more verbose encoding of message passing in Uppaal, but also allows for more control and a possible optimization of the performance. We have not looked in detail into the code optimization yet, but it seems that e.g. the Uppaal CoVer tool [14] may bring a further improvement in the verification performance.

We note that while in TLA$^+$ specification of WS-AT we can count up to 33 rules used in the model, the Uppaal code implements only 22 rules. While still modelling the protocol at exactly the same level of abstraction, the reason is that in the TLA$^+$ language one has to explicitly assert what variables an execution of a rule leaves unchanged. This is due to the co-inductive approach used in TLA$^+$ and user that thinks in imperative programming terms may find it confusing. As Uppaal uses an inductive approach (what is not described in the rules, is not allowed), we eliminate the need to consider rules that have no effect on the protocol behaviour.

Another point we shall discuss is the possibility to extend the approaches with quantitative analysis. The quality of service has recently become an important aspect and one may wish to explicitly model for example time and cost attributes of a protocol. Already the WS-AT specification mentions the time aspects, citing [10]: "A coordination context may have an Expires attribute. This attribute specifies the earliest point in time at which a transaction may be terminated solely due to its length of operation." Both in TLA$^+$ and Uppaal specifications the time-outs are currently modelled using a nondeterministic choice, which on one hand provides a safe over-approximation of the behaviour, but on the other hand does not allow us to ask time related queries. While time features can be specified in both formalisms, TLC does not provide any verification support for it. The analysis of time aspects in Uppaal is straightforward, as Uppaal is a state-of-the-art tool for continuous time modelling and (automatic) verification. Moreover, the Uppaal related tool Uppaal Cora [13] for cost-optimal reachability will allow an easy addition of cost features for analysis and verification of a variety of quality of service questions. For further discussion on this topic the reader may consult also [6].

To sum up, it is possible to verify the consistency of the WS-AT for up to five participants both in TLC and Uppaal. The main problem in the process is actually the understanding of the WS-AT specification which, in its textual form, is incomplete and imprecise. The authors in [4] relied during the formalization phase on two experts that participated in designing the WS-AT protocol. Our modelling task was easier because TLA$^+$ is a fully formal language and we could find all the answers about the behaviour of the protocol in their specification. We can roughly conclude that TLA$^+$ is a more suitable language for higher-level specification of WS protocols because of it succinctness and flexibility. On the other hand, any protocol that is described in Uppaal timed automata framework can be also verified, which is not the case for the TLC model checker. The experimental results also show that the Uppaal engine is noticeably faster than

TLC and hence more suitable for complex protocols. For further applicability in automatic verification UPPAAL provides readily available extensions with time and cost, features that can be encoded in TLA$^+$ specifications but not necessarily verifiable in TLC nor any presently available tool.

The formalization of WS-AT shows the need for introducing a standard for the description of WS protocols. The current practice is insufficient and the standardized protocols can be ambiguous and incomplete. In our future work, we plan to investigate a higher-level language that would share some of the advantages of TLA$^+$ as specification formalism, while being more targeted directly towards WS protocols, easily understandable by software engineers, and allowing an automatic translation to verification tools such as UPPAAL.

Acknowledgments. We would like to thank Kaustuv Chaudhuri, Leslie Lamport and Stephan Merz for answering our questions related to TLC.

References

[1] Behrmann, G., David, A., Larsen, K.G.: A tutorial on UPPAAL. In: Bernardo, M., Corradini, F. (eds.) SFM-RT 2004. LNCS, vol. 3185, pp. 200–236. Springer, Heidelberg (2004)
[2] Gray, J., Reuter, A.: Transaction Processing: Concepts and Techniques. Morgan Kaufmann, San Francisco (1993)
[3] Holzmann, G.J.: Design and Validation of Computer Protocols. Prentice-Hall, Inc., Upper Saddle River (1991)
[4] Johnson, J.E., Langworthy, D.E., Lamport, L., Vogt, F.H.: Formal specification of a web services protocol. Journal of Logic and Algebraic Programming 70(1), 34–52 (2007)
[5] Lamport, L.: Specifying Systems. Addison-Wesley, Reading (2003)
[6] Lamport, L.: Real-time model checking is really simple. In: Borrione, D., Paul, W. (eds.) CHARME 2005. LNCS, vol. 3725, pp. 162–175. Springer, Heidelberg (2005)
[7] Lamport, L., Yu, Y.: TLC — the TLA+ model checker (2003), http://research.microsoft.com/en-us/um/people/lamport/tla/tlc.html
[8] Mathew, B., Juric, M., Sarang, P.: Business Process Execution Language for Web Services, 2nd edn. Packt Publishing (2006)
[9] Newcomer, E., Robinson, I. (chairs): Web services coordination (WS-coordination) version 1.1 (2007), http://docs.oasis-open.org/ws-tx/wstx-wscoor-1.1-specos/wstx-wscoor-1.1-spec-os.html
[10] Newcomer, E., Robinson, I. (chairs): Web services atomic transaction (WS-atomic transaction) version 1.2 (2009), http://docs.oasis-open.org/ws-tx/wstx-wsat-1.2-spec.html
[11] Ravn, A.P., Srba, J., Vighio, S.: UPPAAL model of the WS-AT protocol, Available in the UPPAAL example section at http://www.uppaal.com/
[12] UPPAAL, http://www.uppaal.com
[13] UPPAAL CORA, http://www.cs.aau.dk/~behrmann/cora/
[14] UPPAAL CoVer, http://www.hessel.nu/CoVer/
[15] W3C. SOAP version 1.2 part 0: Primer, 2nd edn, W3C Recommendation (2007)

SPARDL: A Requirement Modeling Language for Periodic Control System

Zheng Wang[1], Jianwen Li[1], Yongxin Zhao[1], Yanxia Qi[2], Geguang Pu[1], Jifeng He[1], and Bin Gu[2]

[1] Shanghai Key Laboratory of Trustworthy Computing
East China Normal University, Shanghai, China, 200062
{wangzheng,lijianwen,yxzhao,ggpu,jifeng}@sei.ecnu.edu.cn
[2] Beijing Institute of Control Engineering, Beijing, China, 100080
qiyanxia369@sina.com, gubin88@yahoo.com.cn

Abstract. This paper develops a requirement modeling language called SPARDL for modeling and analyzing periodic control systems. The system consists of periodic behaviors together with a mode transition mechanism for different behavioral patterns, which is largely applied in the development of control systems of spacecrafts and automobiles. SPARDL can specify the features such as periodic driven behaviors, procedure invocations, timed guard, and mode transition, etc. Each mode in SPARDL can also contain complex activities such as controlling behaviors and data processing. To understand system behaviors precisely, a structural operational semantics is proposed for SPARDL. To analyze periodic control systems in SPARDL, a requirement prototype generation algorithm is proposed to simulate and test the requirements. Meanwhile, a case study is presented to illustrate our approach to requirement modeling and simulation in the development of control systems.

Keywords: Requirement Modeling Language, Control System, Prototype Generation.

1 Introduction

Control systems are widely used in many areas including Automation, Aeronautics etc. One of the most important characteristics for control systems is timed sensitive property. For instance, most of control systems are real-time systems [8,9]. Moreover, one important type of real-time systems is periodic one, which may deal with a series of periodic tasks during some time interval. When a guard is satisfied, or an event arrives, the system enters new state for new periodic tasks until some other guard holds on. The control system of space shuttle is a representative for periodic systems. Requirement capture and analysis for periodic system is a tedious job since there are lots of complex states in the system, and the control and data relations among states may be embedded iteratively. Requirement engineers will spend plenty of time to understand and analyze system requirement manually to find whether there is inconsistency or ambiguity in the requirement.

T. Margaria and B. Steffen (Eds.): ISoLA 2010, Part I, LNCS 6415, pp. 594–608, 2010.

Normally, the periodic control system has the following features:

- Event-driven. Periodic systems have complex states while state transition is driven by kinds of events. For instance, the space shuttle may enter the moon-circling state when some time guard triggered by a timer is satisfied.
- Coherent data. Periodic control systems deal with data collected from environment periodically. The data are processed in each state which may affect other states the system enters. Thus, the data in periodic systems are highly coherent and we need an approach to analysis or testing of data flow in system requirement.
- Fixed tasks. For some state in control system, the tasks performed are fixed in one period. Moreover, some embedded systems require that the tasks can be finished in one period strictly, which can be ensured by the estimation of worst execution time for those tasks under fixed hardware platform.
- Convergent behavior. Periodic control systems may enter some stable state by adjusting its behaviors continually. For instance, Moon-exploration satellite may enter **near-moon-circling** state eventually after it is launched from the earth.
- Hierarchy. Though periodic systems handle kinds of tasks in real time, most of them have distinct hierarchial structure. For instance, those basic tasks which deal with different control algorithms are encapsuled as basic modules while logic control behaviors for the system are composed of basic modules by connectors. As a result, modeling hierarchy is supposed to be one of the goals for any requirement modeling language of periodic systems.

To facilitate the modeling and analysis of periodic control systems, we propose a requirement modeling language called SPARDL, which aims at providing a formal but intuitive modeling mechanism for periodic control systems. As a modeling language for control systems, a SPARDL model consists of a set of modes that are trigged by guards periodically. The behavior in each mode includes three types, mode initialization, frequency procedures and mode transitions. A transition guard in mode may use temporal conditions, which do not only depend on current system state, but also on previous one.

SPARDL has its precise semantics for formal analysis. Requirement engineers would like to check whether there is inconsistency or ambiguity in system requirement. For instance, engineers may be interested in whether unpredicted paths are in the process of mode transition. To analyze the requirement model, an interpreter is designed for SPARDL, which is implemented by transforming from SPARDL model to C code. As a result, a requirement prototype can be generated from SPARDL model automatically. This prototype can simulate the behaviors of system requirement to facilitate engineers to validate system requirement, which is essentially a model-based testing approach. To develop control systems, the system designers specify the whole system behavior by

dynamical and mathematical model. This phase usually consists of modeling the system behaviors, dividing and conquering functionalities from the view of systematic engineering. When the system design is finished, the SPARDL model can be established by system engineers for further implementation. To analyze SPARDL model, a corresponding requirement prototype is generated for simulation and validation.

The rest of this paper is organized as follows. We first give an introduction to SPARDL in Section 2. The semantics of SPARDL is given in Section 3. A simple SPARDL model is used to illustrate the semantics in this section. In Section 4, we develop an algorithm to generate a prototype from SPARDL model. And the generated prototype is implemented in C programming language. The tool implementation is illustrated in Section 5. We make a discussion in Section 6 and give a conclusion in Section 7.

2 The Requirement Modeling Language

This section describes the requirement modeling language SPARDL. The language provides a requirement abstraction for control systems which exhibit period and multi-mode features. The features of SPARDL are similar to those in StateChart [5], where an event can drive the state transition. SPARDL provides rich control structures for the description of control logic compared to StateChart, which focuses on the composition of system states. The low-level elements of this language are similar to those in C language. However, it also provides the mechanism which can describe the behaviors of control systems. Compared with the traditional language, periodic behaviors and timed guard are the main features of SPARDL. Besides that, SPARDL has a two-hierarchy structure for modeling system behaviors. The first one supports to model a control system at the abstraction level, while the second one is used to define the details of each module in the control system.

2.1 Module-Hierarchy Syntax

The syntactic elements in Table 2 support to model the architecture of a control system at the abstract level, while the elements in Table 1 are used to specify the system behaviors in detail.

Table 1. Syntactic Elements in the second hierarchy of SPARDL

$$
\begin{array}{rl}
func =_{df} & id\ stmts \\
stmts =_{df} & stmt \\
 & \mid stmts; stmts \\
 & \mid \textbf{if}\ b\ \textbf{then}\ stmts\ \textbf{else}\ stmts \\
 & \mid \textbf{while}\ b\ \textbf{do}\ stmts \\
 & \mid \textbf{call}\ id \\
stmt =_{df} & x\ :=\ e \\
 & \mid \textbf{skip}
\end{array}
$$

- A function $func$ consists of a unique name id and a sequence of statements $stmts$.
- $stmt$ ranges over primitive statements:
  - $x := e$ is assignment
  - **skip** behaves the same as $x := x$
- $stmts; stmts$ is sequential composition. It firstly executes the first $stmts$. When the first $stmts$ terminates, the second $stmts$ is started to execute.
- **if** b **then** $stmts$ **else** $stmts$ is the conditional construct.
- **while** b **do** $stmts$ is the iteration construct.
- **call** id is the function call site.

2.2 Mode-Hierarchy Syntax

A control system is described as a set of modes, each of which repeats to execute a fixed sequence of procedures. A mode is a triple. The first element $Init$ is a $modu$, and it takes the duty to initialize the mode. The element $Procs$ is a list of $proc$. When the system is in this mode, it sequentially performs the $Procs$ periodically. A $proc$ is a dual tuple. Its first element f is the frequency to execute the $modu$ and $\frac{1}{f}$ is strict to be positive integer. For example, when $f = 1$, the $modu$ will be executed in every period. When $f = \frac{1}{4}$, the $modu$ will be executed once in four periods. The second element $modu$ in a $proc$ is a procedure with a pre-condition. If the pre of a $modu$ is evaluated false in the system execution,, the $modu$ is just skipped.

When a mode finishes executing $modu$s in its $Procs$, its transition guards will be tested to check whether the control system will switch its mode. The third element $Trans$ in a $mode$ is a set of quadruple $tran$. A $tran$ describes the transition from the current mode to another mode. For this purpose, a $tran$ specifies a transition condition $guard$, a unique $priority$ to arbitrate which guard is trigged first when multiple events arrive simultaneously. In general, the third

Table 2. Syntactic Elements in the first hierarchy of SPARDL

$$
\begin{aligned}
Modes &=_{df} \{mode \mid mode = (Init, Procs, Trans)\} \\
Init &=_{df} modu \\
Procs &=_{df} proc; Procs | proc \\
proc &=_{df} (f, modu) \\
modu &=_{df} (pre, stmts) \\
pre &=_{df} b \\
Trans &=_{df} \{tran \mid tran = (guard, priority, modu, mode!)\} \\
guard &=_{df} b \\
&\quad \mid \textbf{after}\ (guard, c) \\
&\quad \mid \textbf{duration}\ (guard, c) \\
&\quad \mid \neg guard \\
&\quad \mid guard \vee guard \\
&\quad \mid guard \wedge guard \\
&\quad \mid guard \rightarrow guard
\end{aligned}
$$

element *modu* converts the values of variables used in the current mode to the ones used by the target mode, or loads those variables with constants. The *mode*! denotes the repetition of the target mode, where symbol ! denotes the recursion of the current state *mode*.

A *guard* is either a boolean expression, or a timed guard. There are two kinds of timed guards in SPARDL: **after** and **duration**. A guard **after**(g, c) holds in a period p if there is another period p' such that the guard g can be satisfied in period p' and it travels c periods from p' to p. A guard **duration**(g, c) holds in a period p if there is the other period p' such that it travels c periods from p' to p and the guard g can be satisfied in each period from p' to p. In Figure 1, the two types of timed guards are illustrated with execution traces, where the dashed arrows are used to denote repetitions in the traces. The state satisfying g is denoted by filled nodes.

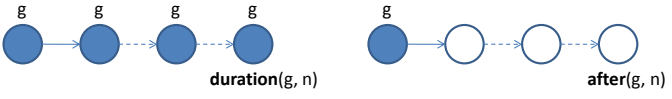


Fig. 1. Timed Guards

Such timed guards provide a mechanism to represent requirements about timed properties. For instance, "When the system is in mode M_1, it should switch to mode M_2 if the angular speed is less than 0.08 rad in four periods". This sentence describes a mode transition from mode M_1 to mode M_2 and the transition condition can be described by this timed guard: **duration**$(a < 0.08, 4)$. To select an element from a tuple $t = (a_1, a_2, \ldots, a_n)$, the projection function is defined as $\pi_i(t) =_{df} a_i, i \in \{1 \ldots n\}$.

3 Operational Semantics for SPARDL

An operational semantics can be expressed as a set of possible transitions which models the system behaviors. The transitions for SPARDL are written as the standard style:

$$\frac{g}{C \xrightarrow{e} C'}$$

where g is the transition condition, e is the event to trigger this transition and C, C' are configurations describing the states of an executing mechanism before and after a transition step. Both g and e are optional.

3.1 Configuration

The configuration for SPARDL is a little complicated for two reasons. First, SPARDL is used to model periodic behaviors. Second, SPARDL supports timed guards. Formally, each configuration is defined as a quadruple:

$$\langle mode!, s, k, Tr \rangle$$

where:

- The first component *mode*! is the current mode to be executed.
- The second component s indicates the current state. And $s \in S$, where S denotes all the possible states of the control system. A state s means a valuation of the variables in the system.
- The third component k records the count of periods for current mode. If a mode transition happens, it will be reset to 0.
- The last component Tr is a trace in which there are sets of atomic boolean expressions. When all the *modu*s in *Procs* are finished, the set of atomic boolean expressions, whose value is true in transition conditions used in $\pi_3(mode)$, is attached to the trace. If a mode transition happens, it will be reset to $\langle\rangle$.

At syntactic level, a *modu* is defined as a *guard* and a sequence of statements. If the *guard* is evaluated to be false, the *statement*s is just skipped. At semantics level, a *modu* is defined as a partial function over S:

$$modu : S \rightharpoonup S$$

The notation $Tr \models g$ means that a timed guard g holds at the trace Tr. The relation $\models$ is recursively defined as below(assuming that g_1 and g_2 are timed guards, e is integer expression and b is boolean expression):

$$\begin{array}{ll}
Tr \models b & \Leftrightarrow b \in last(Tr) \\
Tr \models \neg g_1 & \Leftrightarrow \neg(Tr \models g_1) \\
Tr \models g_1 \rightarrow g_2 & \Leftrightarrow Tr \models \neg g_1 \vee g_2 \\
Tr \models g_1 \vee g_2 & \Leftrightarrow Tr \models g_1 \text{ or } Tr \models g_2 \\
Tr \models g_1 \wedge g_2 & \Leftrightarrow Tr \models g_1 \text{ and } Tr \models g_2 \\
Tr \models \mathsf{after}(g_1, e) & \Leftrightarrow s(e) = c \wedge Tr \models \mathsf{after}(g_1, c) \\
Tr \models \mathsf{after}(g_1, c) & \Leftrightarrow Tr_1 \models \mathsf{after}(g_1, c-1) \\
Tr \models \mathsf{after}(g_1, 0) & \Leftrightarrow Tr \models g_1 \\
Tr \models \mathsf{duration}(g_1, e) & \Leftrightarrow s(e) = c \wedge Tr \models \mathsf{duration}(g_1, c) \\
Tr \models \mathsf{duration}(g_1, c) & \Leftrightarrow Tr \models g_1 \wedge Tr_1 \models \mathsf{duration}(g_1, c-1) \\
Tr \models \mathsf{duration}(g_1, 0) & \Leftrightarrow Tr \models g_1
\end{array}$$

where $c > 0$, $last(Tr)$ means the last element in the trace Tr,
$Tr_1 = Tr[1 \ldots |Tr| - 1]$ and $|Tr|$ means the length of Tr, $s(e)$ means to evaluate the expression e under the state s, which is in the same configuration with the trace Tr.

3.2 Transition Rules

Periodic Behaviors. SPARDL supports the periodic mechanism for control system. In a result, modes will be performed repeatedly and any mode should be performed in particular time points. We assume that there is an operating system to support the control system whose modes will be executed by the OS.

When a period starts, the OS sends a signal to perform the current mode of the control system.

$$\frac{}{\langle mode!, s, k, Tr\rangle \xrightarrow{e} \langle mode; mode!, s, k, Tr\rangle}$$

The event e triggers the current mode to execute. The state of control system remains unchanged.

Mode Initialization. When a mode is performed, its $Init$ element will be firstly executed. The *guard* used in $Init$ is only allowed to be boolean expression.

- If a mode is performed for the first time (the third component in configuration is one, k=1), and the *guard* of *modu* is satisfied, the *modu* is executed. After execution, the $Procs$ of *mode* will be executed either.

$$\frac{pre(s) = true \wedge func(s) = s'}{\langle (Init, Procs, Trans); mode!, s, 1, \langle\rangle\rangle \longrightarrow \langle (Procs, Trans); mode!, s', 1, \langle\rangle\rangle}$$

  where $pre = \pi_1(Init)$ and $func = \pi_2(Init)$
- If a mode is performed for the first time, and the *guard* of *modu* is not satisfied, the *modu* is skipped. And then the $Procs$ of *mode* will be executed.

$$\frac{pre(s) = false}{\langle (Init, Procs, Trans); mode!, s, 1, \langle\rangle\rangle \longrightarrow \langle (Procs, Trans); mode!, s, 1, \langle\rangle\rangle}$$

- If a mode is not performed for the first time, the $Init$ is skipped. And then the $Procs$ of *mode* will be executed.

$$\frac{k \neq 1}{\langle (Init, Procs, Trans); mode!, s, k, Tr\rangle \longrightarrow \langle (Procs, Trans); mode!, s, k, Tr\rangle}$$

Modular Execution. In the modular hierarchy, it is an extension of the traditional imperative language by augmenting subprogram call. For simplification, we do not mention how to compute the semantics of programs in the modular hierarchy deliberately and merely identify each program as state pair. Then we will concentrate ourself on the big-step operational semantics of the mode hierarchy in the following.

When the $Init$ part of a *mode* is finished, the *proc*s in $Procs$ is handled sequentially. The *modu* in a *proc* is executed or skipped based on the frequency $\frac{1}{\omega}$ of the *proc* in each period.

- If the *proc* should not be executed in this period, or the *pre* in *modu* of the *proc* is false, then the *proc* is skipped and the next *proc* is handled.

$$\frac{k\%\omega \neq 0 \vee pre(s) = false}{\langle (proc; Procs, Trans); mode!, s, k, Tr\rangle \longrightarrow \langle (Procs, Trans); mode!, s, k, Tr\rangle}$$

  where $\omega = \frac{1}{\pi_2(proc)}$ and $pre = \pi_1((\pi_1(proc)))$

– If the *proc* should not be executed in this period, or the *pre* in *modu* of the *proc* is false, the element *modu* of *proc* is skipped. If the *proc* is the last one in *Procs*, the element *Trans* of *mode* will be handled next.

$$\frac{k\%\omega \neq 0 \vee pre(s) = false}{\langle (proc, Trans); mode!, s, k, Tr\rangle \longrightarrow \langle Trans; mode!, s, k, Tr\rangle}$$

– If the *proc* should be executed in this period, and the *pre* in *modu* of the *proc* is true, the element *modu* of *proc* is executed and then the next *proc* will be handled.

$$\frac{k\%\omega = 0 \wedge pre(s) = true \wedge func(s) = s'}{\langle (proc; Procs, Trans); mode!, s, k, Tr\rangle \longrightarrow \langle (Procs, Trans); mode!, s', k, Tr\rangle}$$

where $func = \pi_2(\pi_1(proc))$

– If the *proc* should be executed in this period, and the *pre* in *modu* of the *proc* is true, the element *modu* of *proc* is executed. If the *proc* is the last one in *Procs*, the element *Trans* of *mode* will be handled next.

$$\frac{k\%\omega = 0 \wedge pre(s) = true \wedge func(s) = s'}{\langle (proc, Trans); mode!, s, k, Tr\rangle \longrightarrow \langle Trans; mode!, s, k, Tr\rangle}$$

Mode Transition. When the *Procs* has been executed, the *Trans* is performed. If a transition condition is satisfied, then the system switches from current mode to target mode.

– If every *guard* of *tran* is not satisfied, the mode transition does not happen. And the current mode will be executed once again.

$$\frac{\forall tran \in Trans \bullet \neg(Tr_1 \models guard)}{\langle Trans; mode!, s, k, Tr\rangle \longrightarrow \langle mode!, s, k+1, Tr_1\rangle}$$

where $guard = \pi_1(tran)$ and $Tr_1 = Tr^\frown Per(Trans, s)$

The function *Per* associates *Trans* with the set of atomic propositions that are satisfied in state s in the transition conditions.

– If a transition condition, which has highest priority, is satisfied, the system will switch from current mode to its target mode. The periodic count and timed guard trace are all reset.

For simplicity, we let $SU(tran) =_{df} \forall tran' \in Tran \bullet (tran' \neq tran \wedge Tr' \models \pi_1(tran') \Rightarrow \pi_2(tran') < \pi_2(tran))$. If $SU(tran)$ is satisfied if no any other transition condition holds and its priority is higher than *tran*.

$$\frac{\exists tran \in Tran \bullet (Tr_1 \models guard \wedge SU(tran) \wedge pre(s) = false)}{\langle trans; modes!, s, k, Tr\rangle \longrightarrow \langle \pi_4(trans), s, 1, \langle\rangle\rangle}$$

where $guard = \pi_1(tran)$, $Tr_1 = Tr^\frown Per(guard)$, and $pre = \pi_1(\pi_3(tran))$.

– If *pre* is true in a transition, *tran.modu* is executed before the system switches to the target mode.

$$\frac{\exists tran \in Tran \bullet (Tr_1 \models guard \wedge SU(tran) \wedge pre(s) = true \wedge func(s) = s')}{\langle trans; modes!, s, k, Tr\rangle \longrightarrow \langle \pi_4(trans), s', 1, \langle\rangle\rangle}$$

where $func = \pi_2(\pi_3(tran))$.

3.3 A Case Study

We use a simple SPARDL system from Figure 2 to illustrate its usage. This system contains three modes, m_1, m_2 and m_3. Mode m_1 invokes two procedures $proc_{11}$ and $proc_{12}$, with frequencies of 1 and $\frac{1}{4}$, respectively. And this mode can transit to mode m_2 or mode m_3. Similarly, Mode m_2 invokes two procedures $proc_{21}$ and $proc_{22}$ once in each period. This mode can transit to mode m_1 or mode m_3. Mode m_3 invokes procedure $proc_{31}$ with frequency 1. This mode can transit to mode m_1. The system is launched in mode m_1.

We illustrate an execution $C_0, C_0, \ldots$ beginning with the following configurations to explain the semantics of this system in Figure 3.

The execution starts in mode m_1. When the system is triggered by a signal, the modules in mode m_1 begin to perform, $(C_0 \rightarrow C_1)$. And the module $Init_1$ changes the system state s_0 to s_1, $(C_1 \rightarrow C_2)$. Assume that the state s_1 makes the procedure guard g_{11} true, then the procedure $proc_{11}$ changes the system state s_1 to s_2, $(C_2 \rightarrow C_3)$. The next procedure $proc_{12}$ is skipped because its frequency is $\frac{1}{4}$. So the system state s_2 keeps unchanged, $(C_3 \rightarrow C_4)$. When performing the mode transition $Trans$, we assume that the boolean expression $failure$ used in the timed guard evaluates true and the other boolean expression $cmd = AFTERBURN$ is false. Then neither transition conditions are satisfied. The fourth component in the configuration is attached with a new element $\{failure\}$, $(C_4 \rightarrow C_5)$.

When a signal starts to be sent and the system is triggered in a new period, then the modules in mode m_1 start to perform again, $(C_5 \rightarrow C_6)$. Because this mode is executed for the second time, the initialization is skipped and the system state s_2 keeps unchange during the configuration transition $C_6 \rightarrow C_7$. And then the procedure $proc_{11}$ changes the system state s_1 to s_2, $(C_2 \rightarrow C_3)$. The next procedure $proc_{12}$ is skipped again, $(C_8 \rightarrow C_9)$.

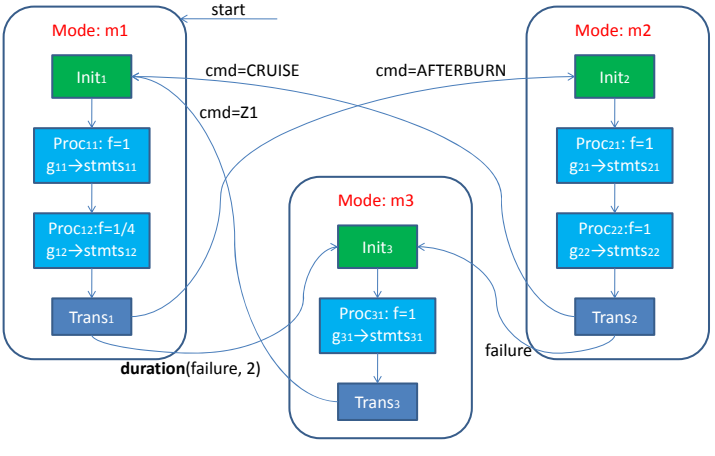


Fig. 2. A Case Study of SPARDL

$$
\begin{aligned}
C_0 &= (m_1!, s_0, 1, \langle\rangle) \\
C_1 &= ((Init_1, (proc_{11}; proc_{12}), Trans_1); m_1!, s_0, 1, \langle\rangle) \\
C_2 &= (((proc_{11}; proc_{12}), Trans_1); m_1!, s_1, 1, \langle\rangle) \\
C_3 &= ((proc_{12}, Trans_1); m_1!, s_2, 1, \langle\rangle) \\
C_4 &= (Trans_1; m_1!, s_2, 1, \langle\rangle) \\
C_5 &= (m_1!, s_2, 2, \langle\{failure\}\rangle) \\
C_6 &= ((Init_1, (proc_{11}; proc_{12}), Trans_1); m_1!, s_2, 2, \langle\{failure\}\rangle) \\
C_7 &= (((proc_{11}; proc_{12}), Trans_1); m_1!, s_2, 2, \langle\{failure\}\rangle) \\
C_8 &= ((proc_{12}, Trans_1); m_1!, s_3, 2, \langle\{failure\}\rangle) \\
C_9 &= (Trans_1; m_1!, s_3, 2, \langle\{failure\}\rangle) \\
C_{10} &= (m_3!, s_4, 1, \langle\rangle) \\
&\ldots
\end{aligned}
$$

Fig. 3. The Configurations of the SPARDL Model

A mode transition happens in $C_9 \rightarrow C_{10}$. In the configuration C_9, the program that remains to execute is the mode transition $Trans_1$, the current system state is s_4 and the Tr is $\langle\{failure\}\rangle$. We assume that the boolean expression $failure$ used in the timed guard evaluates true and the other boolean expression $cmd = AFTERBURN$ is false. Because $Tr \frown \langle\{failure\}\rangle \models \mathsf{duration}(failure, 2)$, the transition condition of the first element in $Trans$ is satisfied. Then the system switches from the current mode m_1 to the target mode m_3. Before the transition, a procedure *action* is executed, which changes the system state s_4 to s_5. The fourth component Tr in the configuration is set to be an empty trace.

4 Requirement Prototype Generation

When the requirement model is established, requirement engineers would like to analyze and validate the requirement to see whether there is inconstancy or ambiguity in it. One analysis approach is to test or simulate the requirement, which means that the requirement model should be executable. One difficulty for making the control system requirement executable is that the control theory for system behaviors is hardly modeled in the requirement since it is a mature physical model involving complicated mathematical equations. The control model is usually provided by control experts in a separate document. Moreover, the implementation of control model is also provided with a separate software package. The basic idea of requirement prototype generation tries to combine the requirement model and control model together to achieve the goal of executable requirement. Figure 4 shows the process of our approach.

Based on the simulator interactions provided by control experts, *simulator protocol* is defined by ourselves. Both *physical environment simulators* and *kinematical computation unit* are provided by control experts. Then, we compile the SPARDL model into C code which can be merged the control package from control community. As a result, the SPARDL model can be executed by the transformation approach which can help the requirement engineers analyze and test the requirement easily.

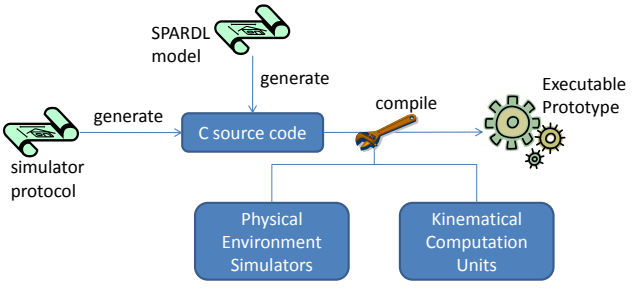


Fig. 4. Requirement Prototype Generation

The SPARDL model contains some features which are mapped to C non-trivially. A mode is transformed into a function. A global variable is introduced to record the current mode. A loop structure is generated to represent the periodic behaviors. The timed guards are implemented through counting. The generated C code introduces a variable to count the number of truth evaluation of the sub-expression in each timed guard. When the count number meets the number specified in this timed guard, the timed guard is satisfied. For the **duration** timed guard, if a false evaluation occurs in a timed guard, the count is reset to 0. When a mode transition happens, all the guard counters are reset 0.

There is a snapshot for generation C code from the SPARDL in Figure 5. A control system is mostly a closed loop system. Firstly, the system gets inputs monitored by sensors. Then the system makes computation based on the input and its internal state. The computation may change its internal state and effect the actuators, which may change the inputs monitored by sensors in the next iteration.

The SPARDL model only focuses on the control and data flow relations but abstracts the kinematical control and environment feedback. Besides the control flow, the execution of a SPARDL model depends on both feedback from the simulation of physical environment and control decision from the computation of kinematical unit. The prototype generation should also consider how to connect the code about the SPARDL model itself with the libraries of those components. We define a protocol to describe the relationship between these read-only/write-only variables in SPARDL model and interfaces provided by those simulators. The glue code is generated based on this protocol. So the generated code should be compiled with libraries of kinematical computation units and physical environment simulators together to generate an executable prototype. Those libraries are implemented by the control experts. In Figure 5, the dashed arrow denotes the function call from the generated C code to the interfaces provided by the *physical environment simulators* and *kinematical computation unit*.

Once the requirement prototype is generated, we can simulate the requirement itself. For instance, we can detect whether some mode is unreachable by testing the requirement. Moreover, the data analysis can also be applied to see what procedures can affect the current active one. The experimental results in the next section shows that our approach is effective in practice.

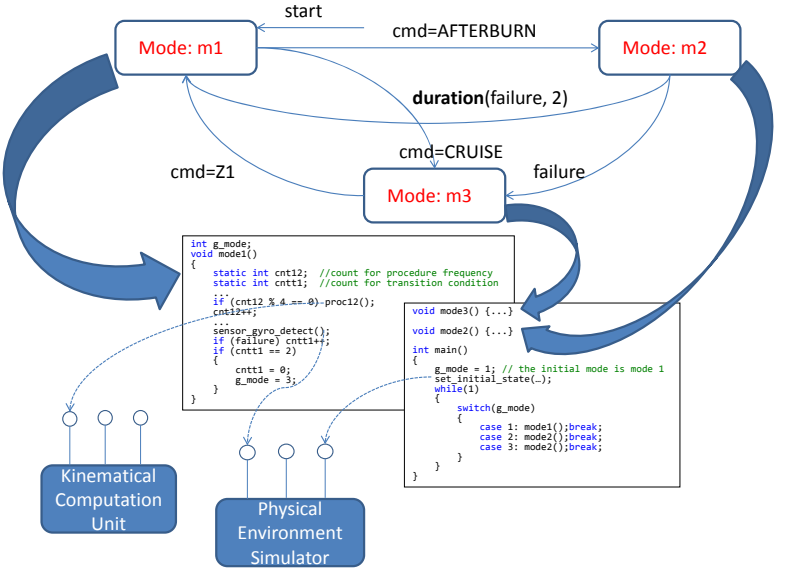


Fig. 5. C Program Generation from SPARDL Model

5 Tool Implementation

We implement our approach in a tool prototype. To help communicate with the requirement engineers, a graphic interface for SPARDL is developed as well. Thus, engineers can just draw the SPARDL model instead of writing SPARDL text, which makes the SPARDL approach be easily accepted by industry. This tool is integrated as an Eclipse plug-in and the snapshot is shown in Figure 6.

The procedure of our SPARDL approach is shown as follows:

1. The control experts provide: (1) corresponding control algorithm implementation, (2) physical environment simulators.
2. The software engineers build a requirement model of SAPRDL and a simulator protocol from the artifacts delivered by control experts.
3. Based on the model and protocol, a requirement prototype is generated automatically. And then the software engineers can simulate the prototype to the test or validate the requirement model.

A real vehicle control case from industry with 10 modes are analyzed by our tool shown in Figure 6. When the vehicle control system is established in SPARDL model, we find two unexpected mode transitions and one may-impossible mode transition. During the simulation, we find that if the control system enters mode m_7, it immediately transits to mode m_6, while it is expected to be remained in mode m_7 at least 600 sec. A similar defect is covered in mode m_4 after the system switches from mode m_4 to mode m_8 too quickly. These three defects are returned to control experts. The former two are confirmed since software

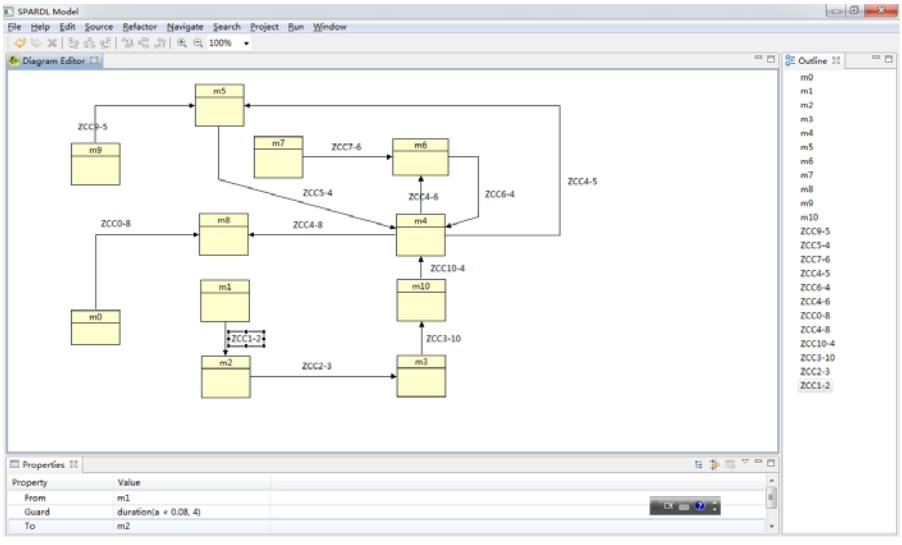


Fig. 6. A Snapshot of Tool Implementation

engineers miss the requirements that the system should remain in those two modes for a time. The third defect is a false alarm. When we use the new test data, the mode transition is taken.

6 Discussion

Our SPARDL model is inspired by the domain-specific language [4] and the executable specification [2] for requirement analysis [6,11]. The DSL Hume [4] provides a powerful modeling suit, such as higher-order functions, polymorphic types, asynchronous communication and exception handling. The same as SPARDL, Architecture description languages (ADLs) [3] also take a macroscopic view for software engineers. Their perspective is shifted from lines of code to modes, tasks and communication among different components. ADLs supports to compile scheduling code to glue different tasks implemented in conventional programming language.

The overall structure of SPARDL model is similar to statechart [5]. In its top level, the model is presented as a statechart. Each mode is described as a state and the mode switches are modeled to be state transition. In statechart, there are two types of state transitions: either a state transition is triggered by an event, or a state transition may happens when a condition satisfies. In SPARDL model, the state transitions in the top level are in the second type. The statechart can be nested while the SPARDL model is also a hierarchy structure. But the sub-structure of a state in statechart is also a statechart while the sub-structure of a mode in SPARDL model is a special activity diagram. Such difference is caused by our motivation. We focus on the mode transition among different modes and

we interest in the behaviors in each mode. So a statechart-like structure is used to describe the mode switches and a activity chart-like structure is introduced to model the behaviors in a mode.

A goal of SPARDL is to provide a concise abstraction for control systems, which is the same with Giotto [7]. Giotto is also a periodic driven mode based modeling language. The tasks in a mode are performed in parallel in Giotto, which is different from SPARDL. Because SPARDL requires the mode transitions happen at the end of modes, while Giotto supports to test mode switch in a mode many times, depending on the frequency of a mode switch. SPARDL extends the semantics of guard by introducing timed guard, while Giotto omits this feature and leaves it to the implementation language. Although both Giotto and SPARDL generate prototypes from models, their purposes are different. The prototype generated from SPARDL is used to validate the functional requirements and will be abandoned after validation.

SPARDL model aims at both capture and validation of system requirement. RCAT [12] introduces a requirement capture notation and checks requirements by converting them into automata. CHARON [1] is a modeling language to describe hierarchical hybrid models and embedded software. Our purpose to generate C code from SPARDL is different from CHARON. We aim at validating requirements instead of generating software implementation automatically. Ptolemy II [10] is a platform to design, model and simulate real-time, embedded systems. It is interesting that the semantics of a model is determined by a software component in the model instead of the framework, which provides a high flexibility.

7 Conclusion

In this paper, we propose a requirement modeling language for periodic control systems. The operational semantics of SPARDL are presented for formal analysis. SPARDL can describe time-triggered periodic behaviors without resorting to times based on the notations such as modes and timed guards we designed. To facilitate the simulation and validation of system requirement, a requirement prototype generation technique is developed. Moreover, the graphical interfaces for SPARDL is developed for the easy communication with system engineers.To facilitate the simulation and validation of system requirement, a requirement prototype generation technique is developed.

Acknowledgements

Geguang PU is partially supported by 973 Project No.2005CB321904, and the Fundamental Research Funds for the Central Universities. Hao XIAO is partially supported by 863 Project No. 2009AA010313. Bin GU is partially supported by NFSC No. 90818024. Jifeng HE is partially supported by the Open Fund of the State Key Laboratory of Software Development Environment under Grant No. SKLSDE-2009KF-2-05, Beijing University of Aeronautics and Astronautics.

References

1. Alur, R., Ivancic, F., Kim, J., Lee, I., Sokolsky, O.: Generating embedded software from hierarchical hybrid models. SIGPLAN Not. 38(7), 171–182 (2003)
2. Anderson, A.H., Shaw, G.A.: Executable requirements and specifications. J. VLSI Signal Process. Syst. 15(1/2), 49–61 (1997)
3. Clements, P.C.: A survey of architecture description languages. In: IWSSD 1996: Proceedings of the 8th International Workshop on Software Specification and Design, p. 16. IEEE Computer Society, Washington (1996)
4. Hammond, K., Michaelson, G.: Hume: a domain-specific language for real-time embedded systems. In: Pfenning, F., Smaragdakis, Y. (eds.) GPCE 2003. LNCS, vol. 2830, pp. 37–56. Springer, Heidelberg (2003)
5. Harel, D.: Statecharts: A visual formalism for complex systems. Sci. Comput. Program. 8(3), 231–274 (1987)
6. Heitmeyer, C.: Using the scr* toolset to specify software requirements. In: WIFT 1998: Proceedings of the Second IEEE Workshop on Industrial Strength Formal Specification Techniques, p. 12. IEEE Computer Society, Washington (1998)
7. Henzinger, T.A., Horowitz, B., Kirsch, C.M.: Giotto: a time-triggered language for embedded programming. Tech. rep., Berkeley, CA, USA (2001)
8. Henzinger, T.A., Sifakis, J.: The embedded systems design challenge. In: Misra, J., Nipkow, T., Sekerinski, E. (eds.) FM 2006. LNCS, vol. 4085, pp. 1–15. Springer, Heidelberg (2006)
9. Kopetz, H.: Real-Time Systems: Design Principles for Distributed Embedded Applications. Kluwer Academic Publishers, Norwell (1997)
10. Liu, X., Xiong, Y., Lee, E.A.: The ptolemy ii framework for visual languages. In: HCC 2001: Proceedings of the IEEE 2001 Symposia on Human Centric Computing Languages and Environments (HCC 2001), p. 50. IEEE Computer Society, Washington (2001)
11. Sadilek, D.A.: Prototyping domain-specific language semantics. In: OOPSLA Companion 2008: Companion to the 23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications, pp. 895–896. ACM, New York (2008)
12. Smith, M., Havelund, K.: Requirements capture with rcat. In: IEEE International Conference on Requirements Engineering, pp. 183–192 (2008)

AutoPA: Automatic Prototyping from Requirements*

Xiaoshan Li[1], Zhiming Liu[2], Martin Schäf[2], and Ling Yin[2,3]

[1] Faculty of Science and Technology, University of Macau
[2] United Nations University - International Institute for Software Technology, Macao
[3] Institute of Software, East China Normal University

Abstract. We present AutoPA, a tool to analyze and validate the consistency and functional correctness of use case designs. The tool directly generates an executable prototype from the requirements. The requirements are captured from different views of the application. Each view is constructed as UML diagram annotated with OCL specifications. Based on a formal semantics, the tool is implemented so that both syntactic and semantic consistency among the provided views can be guaranteed. Afterwards the requirements are analyzed and translated into an executable prototype, allowing the user to interactively validate the functional properties of the requirements model. We illustrate the benefits of the tool using a real-world sized example.

Keywords: Formal Semantics, Requirements Models, Prototyping, Validation.

1 Introduction

In the use case driven incremental development process specifying a system, e.g. by using UML [3,1] and the Rational Unified Development Process (RUP) [7,9], starts with identifying its use cases. Any further step in the process relies on these use cases. Therefore it is crucial that the use cases are consistent.

We propose the use of a formal semantics for use cases which is based on our earlier work on formal semantics of UML models of requirements [11,12]. A model of requirements of an application is defined as a consistent set of models of different views: use case diagrams, conceptual class model, use-case sequence diagrams and data functionality of the use case operations specified as pre- and post-conditions. We extend the set of models with *use case activity diagrams* that specify synchronization of parallel use cases (cf. Section 3.2).

In this paper, we present AutoPA. A tool to automatically generate an executable prototype from a requirements model to check the functional properties and semantic consistency of use cases.

AutoPA provides a rich user interface to design the requirements model of the system but also can work on input from other case tools. The user generates a set of use cases, conceptual class diagrams, and use case activity diagrams. Requirements regarding the data functionality of the system can be provided as OCL specification. The consistency of these views of the system w.r.t. our formal semantics is verified automatically (see Section 3). As a result, we obtain a model of the requirements which has a

* Supported by the projects GAVES funded by Macau Science and Technology Development Fund, NSFC 90718014, 973 program 2009CB320702, and STCSM 08510700300.

T. Margaria and B. Steffen (Eds.): ISoLA 2010, Part I, LNCS 6415, pp. 609–624, 2010.

precisely defined semantics. This model is then automatically translated to an executable prototype (see Section 4). The user can now use the prototype to interactively validate the functional requirements of the system.

We illustrate the functioning of AutoPA on a real-world sized example of a library system (see Section 5) and show how AutoPA can improve the use case identification process.

2 The Library System

Throughout this paper we illustrate the functioning of AutoPA using a running example. Consider a Library System. It maintains a catalogue of three kinds publications: Book, Periodical, and AudioVisual. Periodicals can only be read in the library, while other kind of items can be borrowed. Publications are organized according to subjects for statistic analysis, and each publication can have multiple copies. Users need to be registered with systems, and different policies apply to different types of users, students and teachers. In this paper, we only consider the functionalities handling loans, returns and reservations of users. A class diagram for these functionalities is given in Fig. 1. Invariants specified in OCL can be attached to specify the business rules.

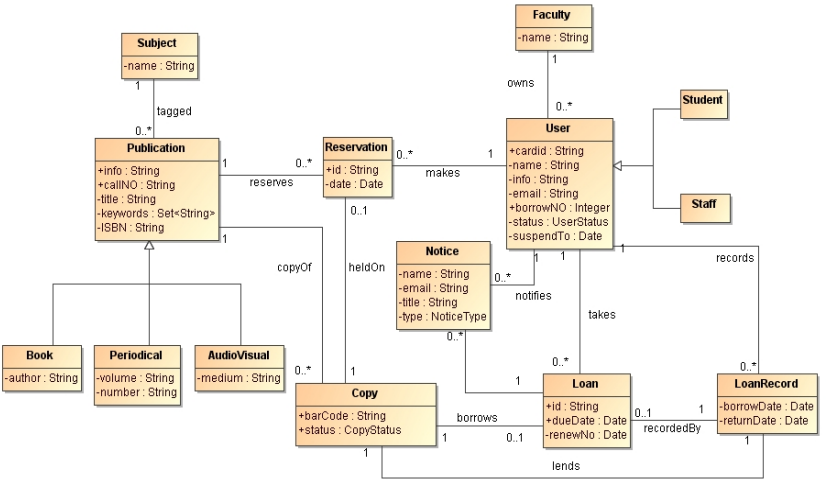


Fig. 1. Conceptual Class Model of the Library System

3 Modeling of requirements

We follow the use-case driven approach to capture requirements that is proposed in RUP [7], which is described in [10,4] more systematically. Informally the process of requirements understanding, capture and analysis is the following:

1. Identify the business processes as use cases, including *actors*, use cases and relations between use cases. Write the description in terms of interactions between actors and use cases, and among use cases, and construct the use case diagram.

2. Identify interaction events and input output data of the system for defining the use case operations and form the activity diagrams.
3. Study and formulate the data functionalities of the use case operations and write their specification in terms of pre- and post-conditions.
4. The previous steps also lead to identification of data types, domain concepts and objects. Based on this, classes and data types are defined, as well as their associations and constraints, and they are represented as a class diagram, called the *conceptual class model* (see Definition the next subsection).

3.1 UML Models of Requirements

Based on the above discussion, we formalize our definition of a model of the requirements.

Definition 1 (Model of requirements). *A model of requirements* $\mathcal{M} = (\mathcal{U}, \mathcal{C})$ *consists of a* use case model $\mathcal{U}$ *and a* conceptual class model $\mathcal{C}$.

Although in the process of requirements capture, identification of use cases may start first, their static functionality and dynamic behavior can only be defined after the data types and classes of objects are defined.

A precise definition of the conceptual class model $\mathcal{C}$ is given in our earlier work [11,12], that we do not repeat here. Informally, the conceptual class model can be represented by a UML class diagram and a set of OCL specifications. The OCL specifications specify constraints on objects and their relations, for example "a Copy that is held for a Reservation must be a Copy of the Publication that the Reservation reserves". Such a class model is called conceptual because its classes does not have methods, and associations do not have direction. Thus, it represents the concepts and their relations in the application domain.

For simplicity, the association name is used as the role name for an end of the association. For example, if p is a Publication instance, $p.reserves$ is the Reservation instance that reserves p; symmetrically if r is an instance of Reservation $r.reserves$ denotes the publication that r reserves.

At the level of requirement modeling, a *state* is the set of existing objects and the relations between them at a distinct point of time during the execution. Thus, the conceptual class model defines the possible *state space of system*, and a state can be represented as a UML object diagram object diagrams. All the object diagrams have to satisfy the invariants of the class model specified in OCL. In later stages of design and implementation, these objects can be refined and implemented by by much more objects. This is also an indicate of the advantage of the tool for scaling up.

3.2 Use Case Model

The use case model includes a use case diagram representing the static relations among the use case cases, an activity diagrams for the dynamic behavior of the use cases, and the specification of the functionality of the use case operations.

Use case diagram. In Fig. 2 we present the use case diagram of the library system. The use case diagram models, for each actor, the use cases that can be executed. This

figure shows one of the main advantages of AutoPA. The translation of the requirements model into an executable prototype gives the user the opportunity to introduce use cases that can test the functionality of other use cases: we introduce a use case *setCurrentDate* with which the *Librarian* actor can do *dailyCheck*. With the *setCurrentTime* operation, a librarian can set the system time of the prototype as the date when a loan becomes overdue and run use case *checkOverdueLoan*. In a real program this use case would usually implemented by an event listener and thus, it would not appear in the specification. By adding a semantic meaning to use cases, a user can actively test her use case diagram by adding other use cases. This leads to a broader and more reliable specification.

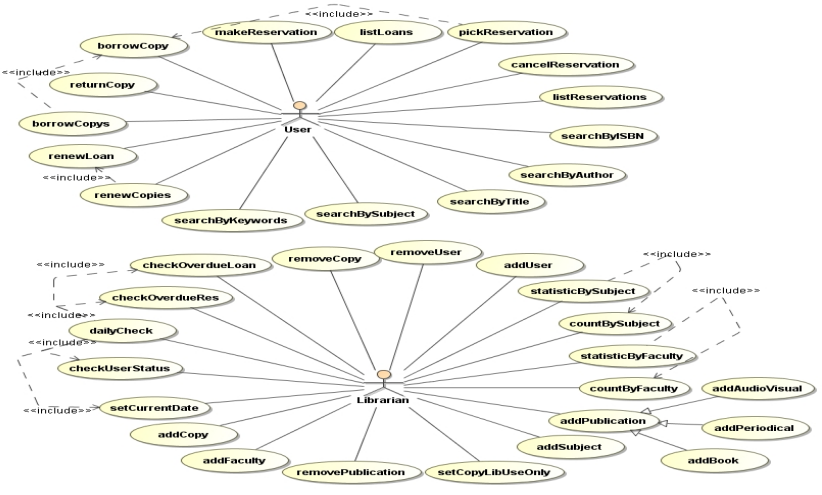


Fig. 2. Use Case Model of the Library System

In AutoPA, the use case diagram is used as the vehicle for creating and integrating (in terms of contexts) the UML models. For the development of AutoPA, we need a formally defined syntax and semantics of use cases. To this end, we adopt the idea of rCOS model of components [4] to use a *use case controller class* to declare a use case[1]. This idea is inspired by the Controller Pattern in [10].

A use case defines a number of *operations* as methods that are to be called by its actor to perform the use case. Use cases can be related in a way that one use is included (or used in the UML terminology) in another.

Definition 2 (Use Case Classes). *A use case class is of the form*

```
Class UseCase Name includes U1,... Uk {
  f: T, ..., f: T;
  m(in: T;out: R){pre: P; post: Q};
  ......
  m(in: T;out: R){pre: P; post: Q};}
}
```

[1] In rCOS, a use case is model as a component with a provided interface and a n interface class is used to ”implement” the interface.

where the keyword "includes" has the same semantics as "extends" in rCOS (or Java), and the types fields f can be data types or classes defined in the conceptual class model, and the pre- and post-conditions are written in OCL,

We can also add use case invariants in the use case class when needed.

Actors and activity diagrams. Actors interact with the system following a protocol in performing a use case. Each interaction event is initiated by the actor by sending an invoking message to the system. This message causes a state change of the system by execution of some a program statement and a return message (possibly with outputs) to the actor.

The *actor class* defines the association between an actor and a use case. For simplicity in the implementation each use case has its own actor named *UseCaseName*User. For example, the actor of *borrowCopies* is named as *borrowCopiesUser*.

```
Class UsercaseNameUser{
  U: UsecaseName;
  activity(){S}
}
```

An actor is similar to a Java thread and its *activity*() is the $run()$ of the thread. The activity method defines the order in which the operations of the use are called and executed to realize the business policy. In rCOS this order is called the *interaction protocol* and defines the dynamic behavior of the use case. The syntax of the body S of *activity*() is given by

$$S ::= m() \mid A.activity() \mid S; S \mid \textbf{if } b \textbf{ then } S \textbf{ else } S \mid \textbf{while } b \textbf{ do } S \mid S \parallel S$$

where $m()$ must be a method of the use case class, and A a name of the actor of a use case class included in the use case class and its meaning can be understood as $A.activity()$ in Java.

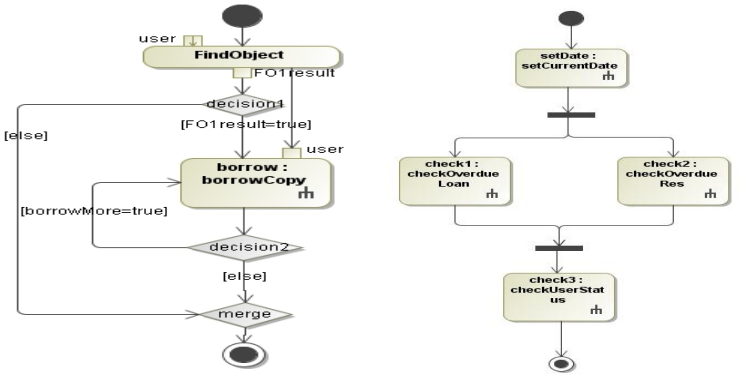


Fig. 3. For the use case *dailyCheck*, fork and join in an activity diagram represents concurrency and synchronization. This use case checks over due loans and over due reservations can start at the same time, and only after both of them finished, use case $checkUserStatus$ can be called.

Note that a method in a use case class or the activity method in an actor do not call methods of the conceptual classes, they are not designed anyway. We use an activity diagram to represent the behavior of the $activity()$ method of an actor. Fig. 3 shows the activity diagrams of two use cases, *borrowCopies* on the left and *dailyCheck* on the right. The diagram on the left shows that use case $borrowCopies$ is carried outs in the following way

1. Input the cardID of the user and locate the corresponding object $user$ in current system state.
2. If the user can be identified, call use case $borrowCopy$ to lend the copy, otherwise, end this use case directly.
3. If user borrows more than one copy, repeat process from Step 2 until no further copy to borrow and then finish this run of the use case.

Definition 3 (Use Case Model). *The use case model of an application is the list of classes for all the use case and actors defined for the application.*

Now with the conceptual class model $\mathcal{C}$ and a use case model $\mathcal{U}$, the operational semantics of each use case operation comes is a transition relation between system states (i.e. object diagrams). It can be directly defined by the interpretation of OCL pre- and post-conditions. The operational semantics of the activity of a use case is then inductively defined from the body statement of the activity method.

4 Design of AutoPA

The AutoPA GUI is designed for creating UML models is designed, but AutoPA also allows its users to use a UML CASE tool, such as *MagicDraw*, to create requirements models of their applications. These models are saved into XMI files. However, the design of the AutoPA GUI and models created by *MagicDraw* must conform to those conditions defined in Section 3. For this, AutoPA defines a meta model, called *UML+OCL* meta model. An instance of this meta model generated from the XMI file is stored in AutoPA and used for syntactic checking.

AutoPA stores the requirements model (e.g., actor and control classes) and the OCL specification separately to allow further editing of the OCL constraints. During the prototype generation, the OCL specifications are translated into a sequence of atomic actions defined below. These actions are needed to avoid potential side effects that may arise when translating pre- and post-conditions into sequential code. Finally an executable prototype is created. The prototype implements the provided requirements model according to our semantics and further provides a graphical user interface allowing the user to trigger different use cases by simply pressing a button.

An architectural overview of AutoPA is shown in Fig. 4. In the following, we describe the used components in more detail.

XMI Parser and OCL Parser. AutoPA XMI file generates an *internal representation of the model* (IRM) from a model of requirements provided by the user (e.g. as XMI

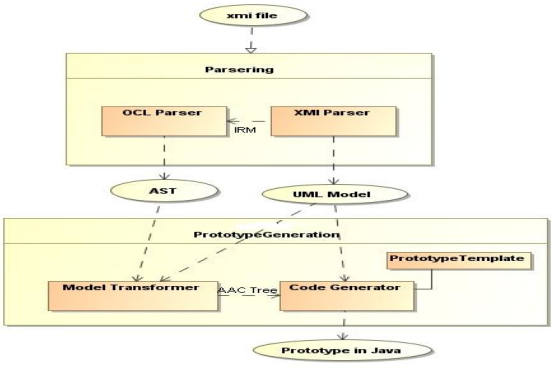


Fig. 4. Architecture of AutoPA with Function Components and Data Flows

file). The IRM includes a *model file*, that records the model elements of the requirements model (classes, associations, actors/control classes, etc.), and a OCL file that specifies constraints on the model elements, and the pre- and post-conditions of the use case operations.

The **OCL Parser** takes generates an Abstract Syntax Trees (ASTs) of the OCL constraint expressions in the IRM. The OCL parser is a reuse of Octopus parser of the OCL validation tool [16].

Model Transformer. takes the IRM file and the ASTs as its input and generates an instance of UML+OCL model. It then transforms each pair of pre- and post-conditions of each use case operation into a sequence of *atomic actions* to implement the state check and change specified by the pre- and post-conditions. For this, AutoPA defines the atomic actions *FindObject*, *FindObjects*, *FindLink*, *FindLinks*, *CheckAttribute*, *CreateObject*, *RemoveObject*, *CreateLink*, *RemoveLink*, and *Update*. Each of them is specified as a pair pre- and post-conditions in OCL and implemented in Java. The first five actions do not change the system state while the other five do. The semantics of these atomic actions are simple and clear from their names. These operations are sufficient to checking conditions and updating a state (the object heap).

In the use case model, the user does not have to write the code for the activity methods. Instead, Model Transformer parses each activity diagram and generates a tree of $ActivityNodes$, from which the Java code of of the activity methods are generated.

Code Generator and Prototype Template. Before generating code, AutoPA first check the well-formedness conditions of models. For this, we define and implement a meta model for the syntax of UML elements with OCL constraints for the models defined in Section 2. After the model passes the well-formedness checking, Code Generator takes the output of Model Transformer and generates the Java code of the prototype. The code include the implementation the conceptual classes, associations relations between these classes, inheritance relations, the use classes, actor classes, and the statements of the use case operations and the activity methods of actor classes.

Prototype Template. is the framework for the prototype GUI and static class declaration. Based on the framework, the Code Generator can generate the prototype java classes from the conceptual class diagram and use case diagram. Finally, the Code Generator generates the prototype GUI that extends the GUI classes of the Prototype Template.

4.1 Implementation of OCL Expressions

OCL is a description language, i.e. pre- and post-conditions are side-effect free. They specify a relation between states (i.e. a next state relation), there is no indication of the orders and combinations of atomic actions for the implementation of the next state relation. We have identified and implemented the patterns between OCL expressions and combinations of atomic actions. For example, Fig. 5 shows the AST of the following OCL expression.

Publication → includes(pub) and Copy → excludes(copy)

By traversing this tree, the *Model Transformer* decomposes the OCL expression into a number of sub-expressions according to the logic structure of expression. Each such sub-expression is an AST fragment that can match one of the atomic action (also specified in OCL). AutoPA defines the rules for matching AST fragments with atomic actions, using AST fragment templates. For example, the atomic action *FindObject (ObjId,Classifier)* checks whether a given instance object with identifier *ObjId* of class *Classifier* exists in the current system state. If it does not exist, the corresponding precondition is false, and the prototype should go to the exception handling. The template **T1** in Fig. 6 and rule **R1** shows a case for generating a *FindObject* action.

$$\textbf{R1} \triangleq \frac{\begin{array}{c} T(fg) = \textbf{T1}, \texttt{OP1} \in \{asSet, asBag, asSequence\}, \\ \texttt{OP2} \in \{includes, notEmpty, one\} \end{array}}{FindObject(V,C)}$$

If *T(fg)* = **T1** and the contents of the fragment parameters satisfy the premises of rule **R1**, the atomic action corresponding to the fragment *fg* is *FindObject(ObjId,Classifier)*.

Fig. 5. Abstract syntax tree of the OCL expression *Publication → includes(pub) and Copy → excludes(copy)*

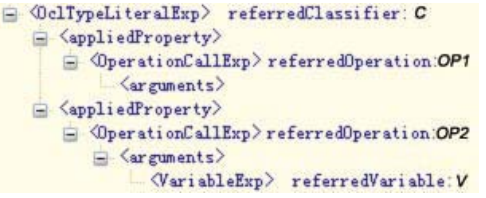


Fig. 6. A template AST template

Execution order of the atomic actions. The OCL expression of a post-condition specifies the change of a state. AutoPA generates the corresponding atomic actions of a post-condition in a correct order. For example, if an expression specifies the removal of an object and the removal of a link to the object, the action for the removal of an object cannot be executed before the actions for removing the link to the object.

AutoPA implements a sorting algorithm when generating the sequence of atomic actions for a pair of pre- and post-conditions. We assign the highest priority to atomic actions *FindObject* and *FindObjects*, and then in a decreasing order for *FindLink*, *FindLinks*, *CheckAttribute*, *CreatObject*, *CreateLink*, *UpdateAttribute*, *RemoveLink* and *RemoveObject*. For the use case *borrowCopy* of the library system, AutoPA generates the following sequence of atomic actions from its specification (that we omit due to page limit).

```
 1. FindObject(u, User)
 2. FindObject(c, Copy)
 3. CheckAttr(u,User, "status","=","normal")
 4. CheckAttr(c, Copy, "status","=","available")
 5. if FindObject(u, Staff)
    then CheckAttr(u, User, "borrowNO", "<", 30)
    else CheckAttr(u, User, "borrowNO", "<", 10)
 6. CreateObject(ln)
 7. Publication pub=FindLink("copyOf", c).getOtherRole()
 8. CreateObject(lr: LoanRecord)
 9. UpdateAttr(c, Copy, "status","=","loan")
10. !FindLink("borrows", ln, c)
11. CreateLink("borrows", ln, c)
12. !FindLink("takes", ln, u)
13. CreateLink("takes", ln, u)
14. !FindLink("recordedBy", lr, ln)
15. CreateLink("recordedBy", lr, ln)
16. !FindLink("lends", lr, c)
17. CreateLink("lends", lr, c)
18. !FindLink("records", lr, u)
19. CreateLink("records", lr, u)
20. UpdateAttr(lr, LoanRecord,"borrowDate",CalendarNow)
21. UpdateAttr(u, User, "borrowNO", u.borrowNO@pre+1)
22. result = true
```

```
23. if Findobject(pub, AudioVisual)
    then UpdateAttr(ln, Loan, "dueDate",CalendarNow+10)
    else if FindObject(u, Staff)
    then  UpdateAttr(ln, Loan, "dueDate",CalendarNow+60)
    else  UpdateAttr(ln, Loan, "dueDate",CalendarNow+30)
```

AutoPA can generate checks for system invariants, which are specified as OCL expressions. System invariants should hold on all system stable states, i.e. In any state before or after a use case operation. Multiplicities in a class diagram are special kinds of invariants. Code for an invariant is generated in the same as for a precondition, it is is in general an iteration statement that checks all objects and links of the current system state.

When generating code from post-conditions, the functionality of AutoPA is limited to post-conditions that are formulated from the primitive *assignable expressions* of the form $e_1 = e_2$, where e_1 is a navigation path $x.a_1.\ldots.a_k$ and e_2 can be evaluated on the current state, i.e the pre-state in term of OCL. To generate code from expressions like $x + y = x@pre + 10$ and $x + 3y = y@pre - 5$ an equation solver or SAT solver is needed. For the same reason, AutoPA does not handle nondeterministic post-conditions such as $x > y@pre + x@pre$.

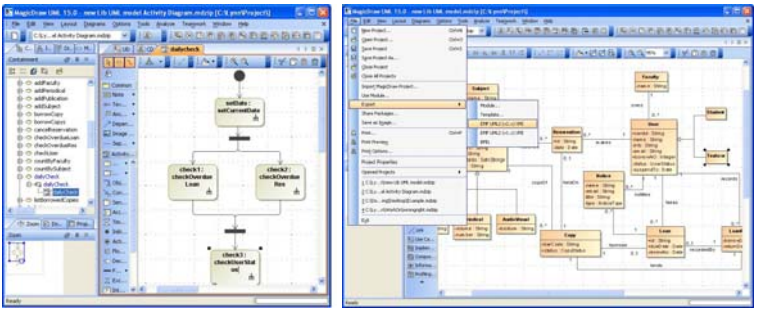


Fig. 7. Specifying a complex use case

5 Prototype of the Library System

We build the requirements model of the library system by drawing diagrams and OCL specifications. The XMI files of this model is then uploaded to AutoPA to generate its executable prototype in Java. We validate the requirements of library system by running the Java prototype, This section demonstrates the results of the generated library prototype system.

We use UML CASE tool MagicDraw to build the requirements model, which includes conceptual class diagram, use case diagram (Fig. 1and 2) and the activity diagrams for complex use case (the left of Fig. 7), and the associated OCL specifications which are attached to use case operations and classes. Finally, the requirements model can be saved the model into an XMI file shown the right of Fig. 7.

5.1 Generating a prototype

We run AutoPA to generate an executable prototype from the XMI file produced from the UML model of requirements in the following steps

1. Run AutoPA and create a new project, see the left of Fig. 8.
2. Import the XMI file produced above, see the right Fig. 8. AutoPA parses it and checks its syntax, and it then reports the result of the syntactic checking as shown in Fig. 9. If the syntactic checking of the XMI file succeeds, AutoPA generates an **IRM** which can be edited in AutoPA, as shown in Fig. 10.
3. Then we generate the prototype from the IRM file imported, as shown Fig. 11.

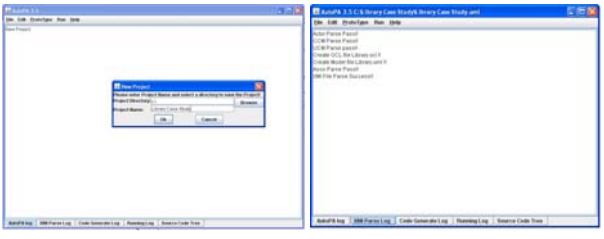


Fig. 8. Create a new project in AutoPA and Import XMI file

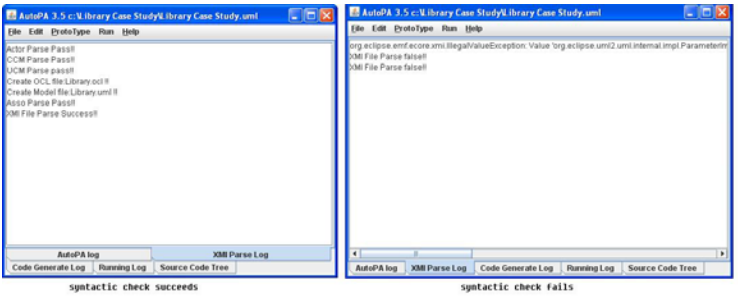


Fig. 9. Results of syntactic check of the XMI file

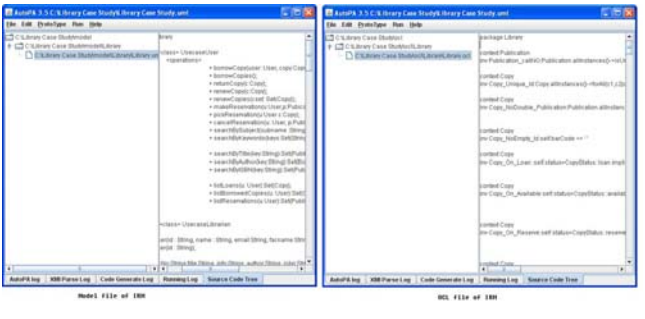


Fig. 10. Editing IRM in AutoPA

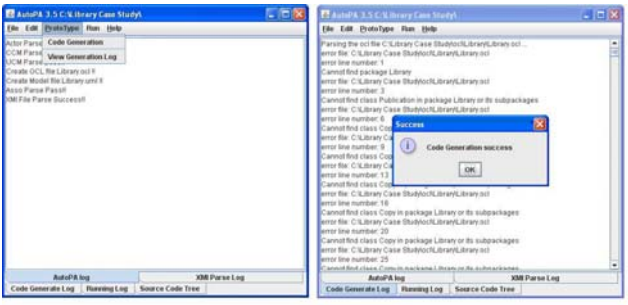


Fig. 11. Prototype generation

5.2 Execution of Generated Prototype

For this case study, the prototype generated by AutoPA contains 156 .java files with a total of 34,522 lines of code. We execute the generated prototype to validate the requirements of the Library system. The interface of the prototype is shown in Fig. 12. Once we select an actor on the left area of the window, the use cases associated with it appears on the right side. For example, in Fig. 12, the actor "Librarian" is selected, and the use cases, "registerUser", "addFaculty" ect., are shown on the right area of the window.

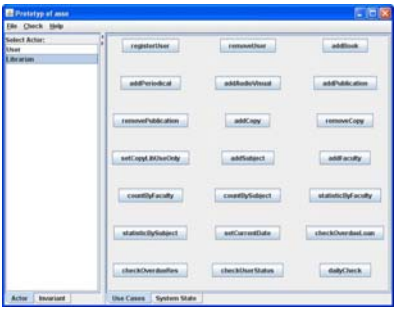


Fig. 12. Interface of generated Library prototype

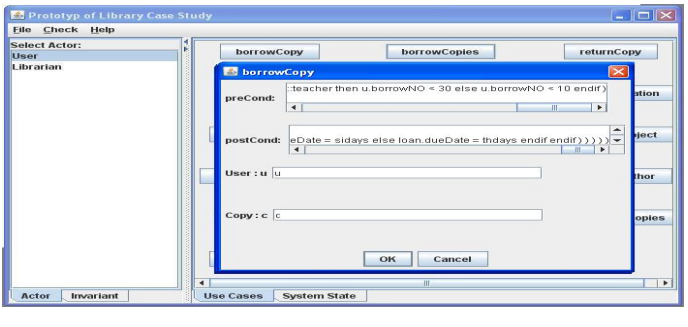


Fig. 13. Interface of use case *borrowCopy*

We can click a use case on the right side and execute it step by step. We run *borrowCopy* use case to illustrate the execution process. Use case *borrowCopy* is specified by a pair of pre- and post-conditions. First, we input the parameters it requires, as shown in Fig. 13, and then execute it by clicking the sequence of atomic actions shown in Fig. 14. A "Green" button means the execution of the atomic action has been finished, a "Yellow" button means the atomic action is to be executed next, a "Red" buttons show the atomic action has not been executed yet. The results of the execution of each atomic action are shown in the *OutPut* area.

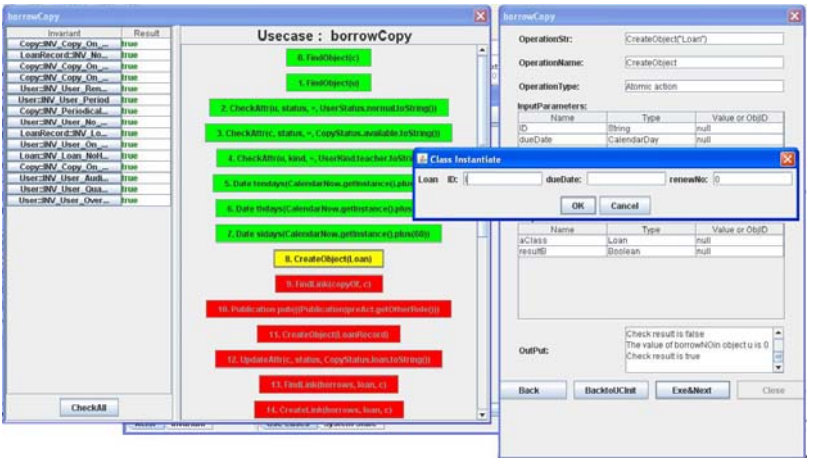


Fig. 14. Prototyping execution of *borrowCopy*

Checking invariants. We can also check multiplicities and invariants on current system state. A screen short of the checking is shown in Fig. 15.

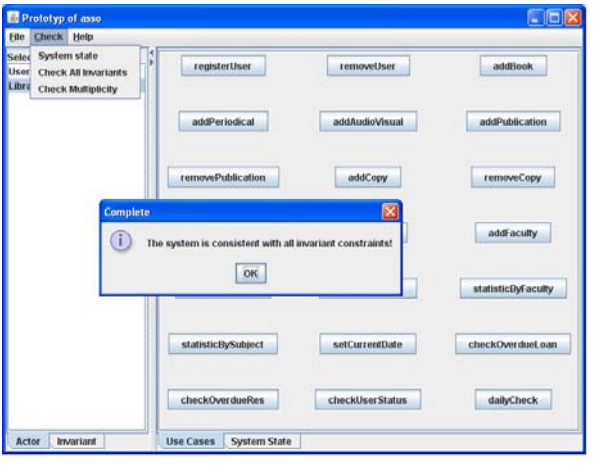


Fig. 15. Screen snaps of checking invariants and its result

```
    public class forAll83 extends ALoopOperation{
        public void accept(List collectionPar) {
            result=true;
            Iterator it=collectionPar.iterator();
            while(it.hasNext()){
                Reservation r=(Reservation) it.next();
                FindLinks a1094= new FindLinks("heldOn", r);
                boolean b1095 = !a1094.getOtherRoles().isEmpty();
                if(!result) break;
            }
        }
    }
public void INV_Copy_On_Available() throws InvariantException
{
    FindLinks a1891;
    FindObjects a1896;
    CheckAttr a1886;
    FindLinks a1888;
    FindLink a1890;
    boolean result = true;
    try{
        a1886 = new CheckAttr(this, "status", "", CopyStatus.available.toString());
                boolean b1887 = a1886.getResult();
        a1888 = new FindLinks("borrows", this);
                boolean b1889 = !a1888.getResult();
        a1890 = new FindLink("copyOf", this);
        a1891 = new FindLinks("reserves", ((Publication)a1890.getOtherRole()));
        a1896 = new FindObjects(a1891.getOtherRoles(), new forAll83());
                boolean b1898 = !a1891.getResult() || a1896.getResult();
                boolean b1899 = b1889 && b1898;
                boolean b1900 = (b1887 ? b1899 : true);
        result = b1900;
    } catch (Exception e) {
        e.printStackTrace();
    }
    if ( ! result ) {
        String message = "invariant Copy_On_Available ";
        message = message + "is broken in object '"+ this.getID() + "' of type '" + this.getClass().getName() + "'";
        throw new InvariantException(this, message);
    }
}
```

Fig. 16. Methods for checking invariant $INV\_Copy\_On\_Available$

An invariant related of a use case is checked after the execution of each atomic action, as shown on the left side of Fig. 14. The methods of checking an invariant are generated as shown in Fig.16. In the example, the method *INV_Copy_On_Available* is generated for checking the invariant *Copy_On_Available* specifying the assertion that a copy is available only if the publication of the copy is not reserved or each reservation of it has a copy held for it but this copy not held for a reservation. This is a complex invariant on class Copy, Reservation, Publication and the associations between them. It is specified as follows and implementation of the checking is shown in Fig. 16.

context Copy
inv Copy_On_Available:self.status=CopyStatus::available
 implies(self.borrows → isEmpty() and
(self.copyOf.reserves → isEmpty() or
 self.copyOf.reserves → forAll(r|r.heldOn → notEmpty()))

6 Conclusion and Discussion

We have presented the formal definition of models of requirements, and the design and implementation of the prototyping tool AutoPA. With the graphical user interface or an existing UML CASE tool, the structural aspects of the models of different views can be prepared, and the use case activities separately by drawing. The model constraints and the pre- and post-conditions of use case operations specified in OCL are prepared in text. The key algorithm of the tool is to translate use case operation in OCL specification into a sequence of atomic actions. The prototype is used for validation functional properties, as well as for syntactic consistency checking.

Comparing with other prototype tools, a distinct feature of AutoPA is prototyping from a model of requirements directly, rather than from a design model, such as design sequence diagrams, a design state diagram, or a model of live sequence charts [15,6]. The tool USE [2] mainly focuses on validating UML models with OCL constraints by testing the given system states (object diagrams) [5]. As discussed in Section 3 and 4, this function is also supported by AutoPA, as constraints on states can be checked in addition to automatic generate of an executable prototype.

Further development of AutoPA includes the OCL *MessageExp* expressions for prototyping real-time interactive systems. We also plan to develop visual animation functionality for demonstrating the dynamic behavior of state changes using the graph-based operational semantics in [8]. Another application could be to use AutoPA for test case generation. This can be realized either by deriving input values from the OCL specification or by introducing special use cases that describe how other use case are tested.

Our experiments with the library system show that AutoPA is a useful tool to support software design by contracts [13,14,4]. AutoPA contributes to the landscape of CASE tools in two ways. 1) By checking the semantic consistency of the use case model we can improve the quality, especially for complex diagrams. 2) The generated prototype allows to validate the functional properties of the system in an interactive way, i.e. the user can experiment with the system.

Acknowledgements. We would like to thank our colleagues Cristiano Bertolini and Volker Stloz for their comments.

References

1. Unified modeling language version 2.0 (2005), http://www.uml.org/
2. Use. a uml-based specification environment (2008), http://www.db.informatik.uni-bremen.de/projects/USE/
3. Booch, G., Rumbaugh, J., Jacobson, I.: The Unified Modelling Language User Guide. Addison-Wesley, Reading (1999)
4. Chen, Z., Liu, Z., Ravn, A.P., Stolz, V., Zhan, N.: Refinement and verification in component-based model-driven design. Sci. Comput. Program. 74(4), 168–196 (2009)
5. Gogolla, M., Büttner, F., Richters, M.: Use: A uml-based specification environment for validating uml and ocl. Sci. Comput. Program. 69(1-3), 27–34 (2007)
6. Harel, D., Marelly, R.: Come, Let's Play, Scenario-Based Programming Using LSCs and the Play-Engine. Springer, Heidelberg (2003)
7. Jacobson, I., Booch, G., Rumbaugh, J.: The Unified Software Development Process. Addison-Wesley, Reading (1999)
8. Ke, W., Liu, Z., Wang, S., Zhao, L.: A graph-based operational semantics of oo programs. In: Breitman, K., Cavalcanti, A. (eds.) ICFEM 2009. LNCS, vol. 5885, pp. 347–366. Springer, Heidelberg (2009)
9. Kruchten, P.: The Rational Unified Process – An Introduction, 2nd edn. Addison-Wesley, Reading (2000)
10. Larman, C.: Applying UML and Patterns. Prentice-Hall International, Englewood Cliffs (2001)
11. Li, X., Liu, Z., He, J.: Formal and use-case driven requirement analysis in uml. In: COMPSAC, pp. 215–224. IEEE Computer Society, Los Alamitos (2001)

12. Liu, Z., He, J., Li, X., Chen, Y.: A relational model for formal object-oriented requirement analysis in uml. In: Dong, J.S., Woodcock, J. (eds.) ICFEM 2003. LNCS, vol. 2885, pp. 641–664. Springer, Heidelberg (2003)
13. Meyer, B.: Object-oriented Software Construction, 2nd edn. Prentice-Hall, Englewood Cliffs (1997)
14. Mitcheel, R., McKim, J.: Design by Conctract by Example. Addison-Wesley, Reading (2002)
15. Plosch, R.: Contracts, Scenarios and Prototypes: An Integrated Approach to High Quality Software. Springer, Heidelberg (2004)
16. Warmer, J.B., Kleppe, A.G.: The object constraint language: getting your models ready for MDA. Addison-Wesley, Reading (2003)

Systematic Model-Based Safety Assessment Via Probabilistic Model Checking

Adriano Gomes[1], Alexandre Mota[1], Augusto Sampaio[1],
Felipe Ferri[2], and Julio Buzzi[2,*]

[1] Centro de Informática, Universidade Federal de Pernambuco
P.O. Box 7458 – Zip 50740-540, Recife, Brazil
{ajog,acm,acas}@cin.ufpe.br
[2] Embraer - Av. Brigadeiro Faria Lima, 2170 – Zip 12227-901, São José dos Campos, Brazil
felipe.ferri@embraer.com.br, julio.buzzi@anac.gov.br

Abstract. Safety assessment is a well-established process for assuring the safety and reliability of critical (aeronautical) systems. It uses probabilistic (quantitative) analysis to provide precise measures about the safety requirements of a system. Traditionally, quantitative safety assessment uses fault-tree analysis, but certification authorities also allow the use of Markov models. In this paper we propose a strategy for quantitative safety assessment based on the Prism model-checker. Prism models are extracted systematically from a high-level model via the application of translation rules. We illustrate our strategy with a representative system design from the airborne industry.

Keywords: Quantitative Safety Assessment, Probabilistic Model-Checking, Prism, Markov Analysis, Aircraft Systems.

1 Introduction

Traditionally, quantitative safety assessment of aircraft systems is based on Fault Tree Analysis (FTA) [3]. This method is frequently used in industrial applications and it is also indicated by certification authorities. The main reason for its practical acceptance is that FTA is conceptually simple and easy to understand [2]. However, certification authorities also accept the use of Markov Analysis [16] to assure safety requirements on the system design.

Both FTA and Markov models use system failure logic information derived from well-known analysis techniques such as Failure Mode and Effect Analysis (FMEA) and Failure Hazard Analysis (FHA) [2]. Based on this information, the analysis methods evaluate the probabilities of the undesired failure conditions to check whether a safety requirement is satisfied or not. Each technique executes this analysis using different mathematical representations; FTA uses static event-based trees and Markov analysis uses stochastic processes. Markov models are more powerful than fault-trees

* Julio Buzzi was an Embraer engineer when this paper was developed. Currently he is a member of the National Civil Aviation Agency - Brazil (ANAC).

T. Margaria and B. Steffen (Eds.): ISoLA 2010, Part I, LNCS 6415, pp. 625–639, 2010.

[2] but they are more complex to be handled, and thus, scarcely adopted in industry. In practice, they are created in a non-systematic *ad hoc* fashion [6, 23]. Despite several automatic model-based approaches for FTA have been proposed [11, 13, 14, 15] using high-level tools like Simulink [9], the treatment of quantitative parameters using FTA still depends on some human intervention. This can introduce errors in the analysis. Moreover, they are not cost-effective, because the probability of each failure condition (top event) must be evaluated singly (just one failure condition at a time), requiring more effort to undertake the analysis of the whole system [1, 17].

In this paper we address these problems proposing a strategy for quantitative safety assessment based on the Prism model-checker [7]. This strategy aims at hiding the interaction with Prism, as well as its Markov-based internal representation (semantics), as much as possible from engineers by providing rules that translate a Simulink diagram, annotated with failure conditions and logic [5, 10], into a Prism model and CSL formulas [8]. We can then check safety requirements, in such a way that we are able to report to the user only those requirements that are not satisfied.

The main contributions of this paper are:

- A (hidden) Markov-based quantitative model-based safety assessment process;
- Translation rules that systematically transform Simulink diagrams (tabular structures), into Prism models augmented with CSL formulas that can automatically verify the quantitative requirements of the system;
- The use of a single model from which it is possible to check any failure condition of a system.

The work presented in [4] defines a methodology that integrates a functional (qualitative) analysis with a non-functional (quantitative) analysis over the system design. In this paper we detail the non-functional analysis strategy, with focus on the automatic model generation and analysis from Simulink diagrams.

The following section presents an overview of (model-based) quantitative safety assessment. In Section 3 we describe our strategy for safety assessment and the rules for the Prism model generation. In Section 4 we show the application of our strategy to a simple aircraft subsystem. Section 5 considers related work and Section 6 shows our conclusions and future work.

2 Overview of Quantitative Safety Assessment

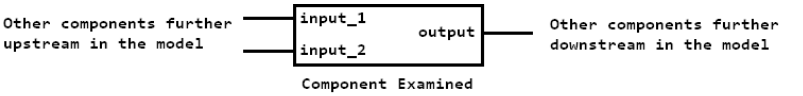
The safety assessment process involves complex phases and activities [1, 2], which aim to minimize the occurrence likelihood of potential hazards. During this process, hazard analysis is performed in parallel with system design. As a result, qualitative and quantitative safety requirements are introduced in the top-level and subsystem design. They comprehend the high-level airplane goals as well as system safety goals that must be considered in the proposed system architectures.

Certification authorities accept FTA, Markov analysis or dependence diagrams as alternatives to perform quantitative safety assessment. The basic information used as input to these techniques are failure conditions and failure rates. Failure conditions are

events of the system (hazards) whose occurrence may lead to a critical situation. They are identified during the FHA analysis, which considers the severity of each failure condition occurrence over the aircraft functions to define the related safety requirement, using an argument (maximum tolerable probability). For example, FHA determines that the probability of occurrence of a catastrophic failure condition must not be greater than 10^{-9} per flight hour. Failure rate is an attribute used to model the likelihood of each basic failure mode (primary and independent failure) of the system. FMEA is a bottom-up method for assessing the failure modes of a system and determining the effects of the relations among these failures. It supplies the failure rates considered in the system. Essentially, a quantitative analysis aims to make reliability predictions for the system. For the certification of an aircraft, it must calculate the average probability of such failure conditions per flight hour, assuming the appropriate exposure time of failures and shows if the results are tolerable.

In the safety-critical systems domain there is an increasing trend towards *model-based safety assessment* [11, 13]. It proposes to extend the existing model-based development activities (simulation, verification, testing and code generation), which are based on a high-level model of the system (expressed in a notation such as Simulink or Statemate), to incorporate safety analyses. These new alternatives are interesting because they are simple, compositional and do not depend on the engineer's skills to be applied. In addition, they can use formal methods, for instance theorem provers, model-checkers and static-checkers [13, 15], to automate, even if partially, the analysis. Moreover, formal methods are one of the alternative methods proposed in DO-178B [12] for the airborne software certification.

Most model-based strategies for safety assessment are mainly based on FHA and FMEA [11] (and in particular on its newer variant, IF-FMEA --- Interface-Focused FMEA [5, 10]). IF-FMEA is of particular interest because it uses a hierarchical tabular structure (see Fig. 1) very useful to capture the transformation and propagation of failures in a system, allowing that complex systems be modeled in a compositional way. Considering the identified failure conditions, and their tolerable rates, during the FHA analysis they are also included in a tabular structure [2, 5], which can be easily incorporated into a design tool like Simulink, using annotations.



Output Failure Mode	Description	Input Deviation Logic	Component Malfunction	λ(f/h)
Omission-output	The component fails to generate the output	Omission-input_1 AND Omission-input_2	Jammed OR Short-circuited	$5x10^{-7}$ $6x10^{-6}$
Wrong-output	The component generates wrong output	Wrong-input_1 OR Wrong-input_2	Biased	$6x10^{-8}$
Early-output	Early output	...	...	...

Fig. 1. IF-FMEA of a hypothetical component system

The table shown in Fig. 1 records four pieces of failure based information and a descriptive field: (i) the possible failure modes of a component; (ii) the dependency of such failure modes with respect to the identified failures via their input ports; (iii) what happens upon a certain failure mode occurrence and (iv) a failure rate.

2.1 Overview of Prism towards Safety Analysis

Prism [7, 8] is a formal probabilistic analysis tool that enables the analysis of Markov models specified in discrete time (DTMC), continuous time (CTMC), and Markov decision processes (MDP). Modules and variables are the basic components of the language and the system is built from the parallel composition of a set of modules. Modules can interact with each other (synchronization) and contain a number of variables that reflect their possible states. The behavior of a module is determined by a list of guarded commands. Each command (initiated by a [], possibly with a label inside) is formed of a guard (boolean expression before the symbol ->) followed by a rate or probability (where 1 means 100%) based transition (where dashed decorated state variables are assigned values, standing for a state update). The transitions represent which state changes are possible and how often they occur.

Fig. 2 illustrates a Prism specification of two modules. Its Markov representation is shown at the right-hand side of this figure. The first line of this specification states that we are considering a continuous time Markov chain that is composed of a set of discrete states, where each of them is the representation of the state (operational, degraded and faulty) of each failure mode (local variables) of a component. This chain of events requires the use of exponential probability distributions for modeling failure mode rates and repairs (this is why we use the CTMC model).

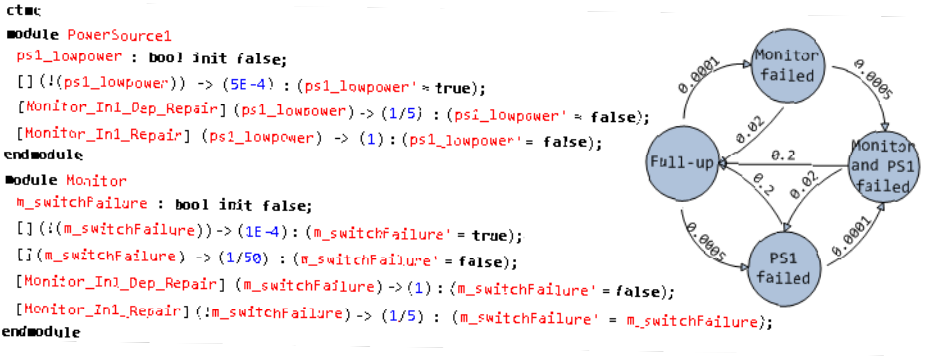


Fig. 2. System representation using Prism

The first module, `PowerSource1`, specifies an abstract failure behavior of a power source. The variable `ps1_lowpower` represents its single failure mode. The first transition captures one of the possible changes in the failure mode: from an operational state it can fail with a rate of $5e^{-4}$ (failure/hour). The next two commands represent repair transitions. These commands are synchronized (the labels inside [and] state the synchronization points) with the module `Monitor`. They work similarly to the first transition of this module, except that they need to synchronize with the corresponding

labels of the module `Monitor`, allowing them to be triggered. The module `Monitor` also uses a single variable: `m_switchFailure`. Its first command states a failure transition command whereas the second represents the capability of its single failure mode being repaired with a rate of 1/50 (repair/hour). The last two commands represent repair transitions corresponding to the repair transitions of the `PowerSource1`.

To analyze the failure behavior of this system, we can use, depending on the purpose, a steady-state or transient analysis [16]. Transient analysis represents the instantaneous failure rate over a single period T whereas the steady-state analysis approximates the long-term average failure rate over multiple time intervals T. The choice over these types of analyses depends on how system repairs are handled. Transient analysis can be performed in either closed-loop (models with repairs) or open-loop models (models without repairs), whereas the steady-state analysis can be performed only on closed-loop models.

Fig. 2 shows that our proposed model considers repair transitions as if they occurred at constant rates, thus it is a typical closed-loop model and both analyses can be performed. We calculate the average rate of a failure condition applying the steady-state analysis. Particularly, the steady-state analysis provides adequate accuracy on their results for certification purposes, since as well as the model shown in Fig. 2, most critical systems are modeled in such a way that they can deal with latency. In this scenario, several components affecting the system functionality must be monitored, maintained at regular intervals and repaired if they are faulty. Although the transient analysis is "exact", it applies strictly to just a single interval T, as if this was the entire life of the system, whereas most critical systems have maintenance cycles, where are periodically restored to the full-up condition. Hence, the analysis more representative for this scenario is when the period T usually represents a repetitive repair interval rather than a life limit [16].

To perform all this quantitative safety analysis Prism uses the CSL language [8]. The operators S (steady-state) and P (transient) of Prism are used to reason about the tolerable probabilities of all system failure conditions. For example, with the formula:

$$S <= 10^{-9} [\text{“Failure Condition”}] . \quad (1)$$

we can check if, in the long run, the probability that a certain "`Failure Condition`" can occur is less than or equal to 10^{-9}. Note that the evaluation of such an expression yields "yes" or "no", based on the corresponding quantitative analysis (the value is implicit). We can also check the exact probability itself by using another CSL formula

$$P = ? [\textbf{true} \ U<=3600 \ \text{“Failure Condition”}]. \quad (2)$$

This yields the instantaneous probability of occurrence of a certain "`Failure Condition`" within 3600 time units.

3 Proposed Strategy

In this section we present our strategy to perform quantitative safety assessment using the Prism model-checker [7]. Fig. 3 presents an overview of our strategy. It starts by collecting the system description, which contains the system block diagrams and a

failure logic model. With this information, we apply our translation rules to create a Prism specification and the associated CSL formulas to analyze the safety requirements of this specification. Then, the Prism model-checker is invoked to check all formulas and only when one of them is not satisfied, this is reported to the user.

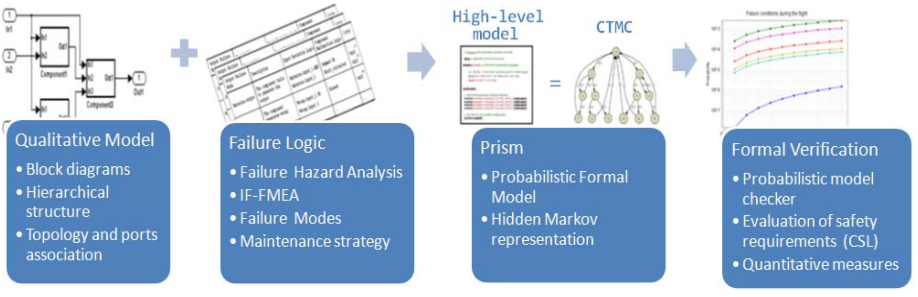


Fig. 3. Overview of the proposed strategy

3.1.1 Input Data Model

Although the annotations that we use in the Simulink diagram are similar to the tabular structures presented during the safety analysis, our rules are stated in terms of the abstract syntax presented in Fig. 4. These data structures are an abstract representation of all the information introduced previously (see Section 2).

```
System              ::=  System_Name X Seq(Subsystem)
Subsystem           ::=  System | Module
Module              ::=  Module_Name X Type X Seq(Deviation) X Seq(Malfunction)
                         X Seq(Port) X MaintenanceStrategy X InspectionTime
Port                ::=  Port_ID X AssociatedPort
Deviation           ::=  Deviation_Name X Port_ID X Annotation X Criticality
Malfunction         ::=  Malfunction_Name X Rate X Annotation
MaintenanceStrategy ::=  MS_Type X AssociatedPort
MType               ::=  Self-Monitored | Monitored | Unmonitored | Monitor
Port_ID             ::=  In<<IN >> | Out <<IN>>
AssociatedPort      ::=  Module_Name X  Port_ID | empty
Annotation          ::=  empty | Malfuntion_Name | Deviation_Name X Port_ID
                         | And <<Annotation1 , Annotation2>>
                         | Or <<Annotation1, Annotation2>>
Criticality         ::= IR | Empty
InspectionTime      ::= IR
Rate                ::= IR
```

Fig. 4. Abstract syntax based on tabular annotations

We start by considering a system (`System`) as a structure that contains a name (`System_Name`) and a list of subsystems (`Seq(Subsystem)`). Each subsystem can be another system or a module, because components can also be systems. A module (`Module`) represents the lower level component that has a name, a list of ports (`Seq(Ports)`), a list of deviations (`Seq(Deviation)`), a list of malfunctions (`Seq(Malfunction)`), the maintenance strategy info and the inspection time. All

these elements are associated with the tabular structures used to store all system information about its architecture, hierarchy, failure conditions, failure modes, repairs and the characteristics of monitoring and propagation of component failures. `Port` is a structure that contains a `Port_ID` (representing the identifiers of input/output ports) and an `AssociatedPort` (which stores the connected port of another component).

`Annotation` is a boolean expression that represents the failure logic of deviations. Its definition considers And/Or operators and their terminal terms can be malfunction names or deviations from any port. `Criticality` represents a real number ($\mathbb{R}$) used to quantify the tolerable probability associated with a failure condition (expressed via a deviation). Finally, `InspectionTime` and `Rate` are also real numbers used to represent the rate of occurrence of a malfunction and of a repair, respectively.

3.2 Translation Rules

Our strategy applies a set of translation rules that are based on the abstract syntax of Fig. 4. To ease the overall understanding of their applicability we also provide the typical sequence of their application in Fig. 5.

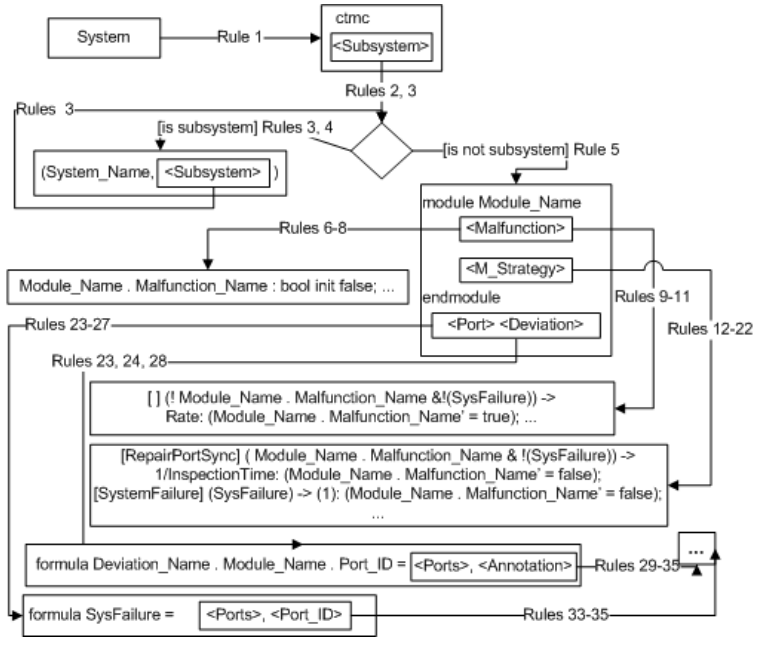


Fig. 5. Translation Strategy Overview

The strategy always starts by applying Rule 1, which states that we are dealing with a CTMC Markov model and applies other rules to create the several Prism modules from the system components (Rules 2-4). The body of a module is effectively created by Rule 5. After that, basic declaration instructions (Rules 6-8), commands (Rules 9-11) and repair transitions (12-22) are created. To complete the translation

strategy, formula expressions are created (Rules 23-28) using a set of rules that decomposes all logic expressions (Rules 29-35). Some rules are omitted because they are very similar to others presented. For instance, Rules 6 and 7 are missing because they are similar to Rules 2 and 3.

3.2.1 Compound Systems and Subsystems

Our rules are inductively defined on the structure of a Prism system. We start with Rule 1 that takes as argument a pair where the first element has the name of a system (`SName`) and the second element a list of its subsystems (`SubSys`).

Rule 1 $|\{$ `(SName, SubSys)` $\}|^{system} \Rightarrow$ ***ctmc*** $|\{$ `SubSys` $\}|^{subsystem}$

Following Rule 1, the resulting Prism code is basically the directive ***ctmc*** (instructing Prism to perform a CTMC interpretation), and a call to the function *subsystem*. This function is defined by Rules 2 (base case) and 3 (recursive case).

Rule 2 $|\{$ `<S>` $\}|^{subsystem} \Rightarrow |\{$ `S` $\}|^{module}$

Rule 3 $|\{$ `S: tail` $\}|^{subsystem} \Rightarrow |\{$ `S` $\}|^{module} |\{$ `tail` $\}|^{subsystem}$

Rules 2 and 3 do not produce Prism code. They access each component of this system and call the function *module* recursively for each component (Rules 4 and 5).

3.2.2 Module

As modules can be subsystems as well, we translate modules by using two rules: Rule 4 (which calls function *subsystem*) and Rule 5 (which starts the creation of a Prism module).

Rule 4 $|\{$ `(SName, SubSys)` $\}|^{module} \Rightarrow |\{$ `SubSys` $\}|^{subsystem}$

Rule 5 takes as input a tuple containing the module elements: name, type, set of ports, set of deviation logics, malfunctions, maintenance strategy and inspection time. The module name (`MName`) is used to name the Prism module (between the keywords ***module*** and ***endmodule***). Inside the module, the function *declars* is called to create the declaration part, and the next two functions the behavioral part. Finally, the function *formulas* is called to create the set of Prism formulas outside the module.

Rule 5 $|\{$ `(MName,Type,Ports,Deviations,Malfuncs,MStrategy,IT)` $\}|^{module} \Rightarrow$

```
module MName
      |{ MName, Malfuncs }|^declars
            |{ MName, Ports, Malfuncs }|^failureCommands
            |{ MName, Ports, Malfuncs, MStrategy, IT }|^repairCommands
endmodule
|{  MName, Ports, Deviations, true }|^formulas
```

3.2.3 Declarations

Malfunctions are representations of possible failures within a component. To capture this feature in Prism, for each component malfunction, local boolean variables initialised with *false* are defined.

Rule 8 |{ MName, (MfName, Rate, Annot) }|^{declar} $\Rightarrow$
MName . MfName: ***bool init false***;

Rule 8 uses each component malfunction to generate the declaration of its respective local variable inside the module block. Module Name (MName) and Malfunction Name (MfName) are used to create the local variable name.

3.2.4 Failure Transition Commands

Rule 11 translates each malfunction into a Prism command. It always assumes the guard as a logical conjunction between the negation of a malfunction (this comes from Rule 8) and the negation of the fully failed system situation (a term defined by a Prism formula). If such a guard is valid then, with a rate given by Rate, this malfunction is activated.

Rule 11 |{ MName, Ports,(MfName, Rate, Annot) }|^{command} $\Rightarrow$
[] (!(MName .MfName**) & !(SysFailure)) ->** Rate: **(**MName .MfName**'=*true*);**

3.2.5 Repair Transition Commands

Rules 12 through 17 translate the maintenance strategy (defined for each component) into Prism repair commands. This is performed according to the classification of each basic component of the system with respect to the treatment of the type of monitoring of its faults. Rule 12 considers two types: Self-monitored and Unmonitored (note the **provided** clause), whereas Rules 13 and 14 tackle the other cases: Monitored and Monitor, respectively.

In Rule 12, if the corresponding guard is valid, then, with a rate (1/Inspection Time), all component malfunctions are deactivated. Function *orLogic* takes a logical disjunction between all malfunctions (this comes from Rule 8) and function *Update* deactivates all malfunctions (set the value *false* to each malfunction).

Rule 12 |{ MName,Ports,Malfuncs,(MSType,AssocPort),IT}|^{repairCommands} $\Rightarrow$
[] ((|{ MName, Malfuncs }|^{orLogic} **) & !(SysFailure)) -> (1/IT):**
|{ MName, Malfuncs }|^{update} **;**
[SystemFailure] (SysFailure) -> (1): |{ MName,Malfuncs }|^{update} **;**

provided MSType = Self-Monitored or MSType = Unmonitored

Rule 13 |{ MName,Ports,Malfuncs,(MSType,AssocPort),IT }|^{repairCommands} $\Rightarrow$
|{ Malfuncs, AssocPort, IT }|$^{\text{monitoredRCommand}}$
[SystemFailure] (SysFailure) -> (1): |{ MName, Malfuncs}|$^{\text{update}}$ **;**

provided MSType = Monitored and AssocPort ≠ empty

However, if the component is Monitored, its repair commands must be synchronized with the Monitor component (function *monitoredRCommmand*).

If a component is a Monitor, instead of the synchronized repair commands corresponding to the monitored component (function *sincronizedRCommand*), another repair command is created to represent the single repair of this component.

Rule 14 |{ MName,Ports,Malfuncs,(MSType,AssocPort),IT }|^{repairCommands} ⇒
[] ((|{ MName, Malfuncs }|^{orLogic}) **& !(SysFailure)) -> (1/IT):**
|{ MName,Malfuncs}|^{update} **;** |{ MName,Malfuncs,Ports,IT}|^{sincronizedRCommand}
[SystemFailure] (SysFailure) -> (1): |{ MName, Malfuncs }|^{update} **;**

provided MSType = Monitor

Rules 15 and 18 are used to define the synchronized repair commands between the monitored (Rule 15) and the monitoring component (Rule 18).

Rule 15 |{ Malfuncs, (MName, PortID), IT }|^{monitoredRCommand} ⇒
[MName . PortID . DependentRepair] ((|{ MName, Malfuncs }|^{orLogic} **) &**
!(SysFailure)) -> (1/IT): |{ MName, Malfuncs }|^{update} **;**
[MName . PortID . Repair] (|{ MName, Malfuncs }|^{orLogic} **) -> (1):**
|{ Malfuncs }|^{update} **;**

Rule 18 |{ MName,Malfuncs,(Port_ID,AssocPort),IT }|^{sincronizedRCommand} ⇒
[MName . PortID . Repair] ((|{ MName,Malfuncs }|^{OrLogic} **) &**
!(SysFailure)) -> (1/IT): |{ MName,Malfuncs }|^{update} **;**
[MName . PortID . DependentRepair] ((|{ MName,Malfuncs }|^{orLogic} **)**
-> (1): |{ MName,Malfuncs }|^{update} **;**

3.2.6 Formulas

The final elements we address are Prism formulas. They correspond to the failure logic expressions annotated in Simulink diagrams. Rule 28 creates such formulas compounding a name for the formula based on the deviation name (DName), followed by the module name (MName) and the identifier of the port (PortID). The formula's body is a boolean expression resulting from function *fExpression*.

Rule 28 |{ MName, Ports,(DName, PortID, Annot) }|^{formula} ⇒
formula DName . MName . PortID = |{ Ports, Annot }|^{fExpression}

The entire set of rules can be found in [19]. Using these translation rules, we generate a valid formal failure model retaining the semantics of diagrams and the system hierarchical model.

3.3 Modeling Considerations

Our solution still does not consider bidirectional data flows (such as the propagation of failure as short-circuit).Yet, such features can be added by considering new translation rules. Our strategy is sound with respect to the following assumptions:

- Component failures are detected in flight only and repaired during ground maintenance or before the next flight (description level), but the failures and repairs occur at constant rates (model level).
- The system is assumed with perfect failure coverage and can to reconfigure to a degradable mode within no time.

In terms of completeness, our rules are complete in the sense that they can translate any Simulink diagram annotated with failure logic in the IF-FMEA style [5]. Besides, this approach is not limited to just using the Simulink diagram as input. Actually, the

necessary input data, which contains information from the qualitative model and the respective failure logic and propagation, is obtained from the tabular structures, which are user defined. Simulink diagrams work implicitly with these structures [4, 10].

Our strategy follows a systematic process that has proved viable and of little impact in practice, since the tabular structures are generated by traditional methods and analyses used by the aircraft industry during the qualitative safety assessment (FHA, FMEA, IF-FMEA, CCA). So, adding a plug-in to some usual design tool, it is possible to automate our systematic approach.

The primary limitation of a stochastic model-checking is the size of the reachable state space, though recent breakthroughs allow very large ($> 10^7$ reachable states) state spaces to be explored in reasonable time.

4 Case Study

Our case study is the Actuator Control System (ACS) (see Fig. 6). Its function is to control the displacement of an electrical actuator. Although it is a simple example, it is representative in the aeronautics context in the sense that it has dependent and independent failures, a hierarchical architecture, latency, evident, repeated and developed events [1, 2]. Considering the Simulink diagram of Fig. 6, annotated with the corresponding failure logic, we can generate the formal specification (see [19] for the complete failure logic of the system), which is depicted in Fig. 7.

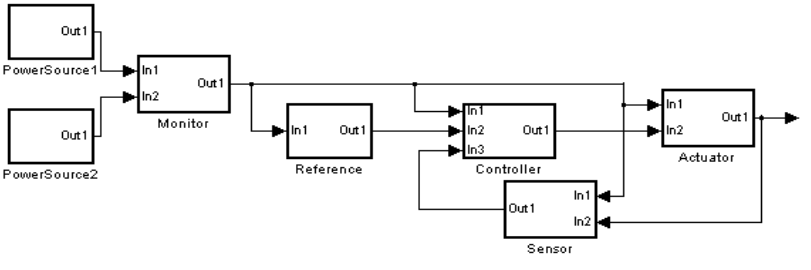


Fig. 6. Actuator Control System

The first two modules of the generated Prism model have already been explained in Fig. 2. The module `PowerSource2` is very similar to `PowerSource1`, and is omitted. The module `Sensor` contains two local variables that represent its failure modes. For each failure mode we have a command to represent its failure transition. We use a single repair transition to update both failure modes to operational states. The module `Actuator` contains several local variables and each failure transition is defined considering its corresponding failure rate. Its repair transition considers the repair rate defined for this component. At the end, formulas are defined to capture failure propagation via module outputs. The remaining modules (`Reference` and `Controller`) are similar and were omitted for conciseness.

```
ctmc
module PowerSource1 ... endmodule
module Monitor ... endmodule
...
module Sensor
 sensor_sensorfailure : bool init false;
 sensor_sensordegradation : bool init false;
 [](!sensor_sensorfailure & !(SystemFailure) ) -> (5E-4) : (sensor_sensorfailure' = true);
 [](!sensor_sensordegradation & !(SystemFailure) ) -> (5e-4) : (sensor_sensordegradation' = true);
 [] ((sensor_sensorfailure | sensor_sensordegradation) & !(SystemFailure) ) -> (1/5)
 : (sensor_sensorfailure' = false) & (sensor_sensordegradation' = false);
 [SystemFailure](SystemFailure)-> (1): (sensor_sensorfailure'=false)&(sensor_sensordegradation'=false);
endmodule
formula CorruptedSignal_Sensor_Out1 = sensor_sensordegradation;
module Actuador
 actuador_lossofdriver : bool init false;
 actuador_lossofmotor : bool init false;
 actuador_mechanismjamming : bool init false;
 actuador_mechanismdegradation : bool init false;
 actuador_driverdegradation : bool init false;
 [](!actuador_lossofdriver & !(SystemFailure) ) -> (1E-4) : (actuador_lossofdriver' = true);
 [](!actuador_lossofmotor & !(SystemFailure) ) -> (1E-3) : (actuador_lossofmotor' = true);
 [](!actuador_mechanismjamming & !(SystemFailure) ) -> (1E-3) : (actuador_mechanismjamming' = true);
 [](!actuador_mechanismdegradation & !(SystemFailure))->(1.5E-3):(actuador_mechanismdegradation'= true);
 [](!actuador_driverdegradation & !(SystemFailure) ) -> (8E-5) : (actuador_driverdegradation' = true);
 [] ((actuador_lossofdriver | actuador_lossofmotor | actuador_mechanismjamming |
     actuador_mechanismdegradation | actuador_driverdegradation)  & !(SystemFailure) ) -> (1/5) :
     (actuador_lossofdriver'= false)&(actuador_lossofmotor'= false)&(actuador_mechanismjamming'= false)
     & (actuador_mechanismdegradation' = false) & (actuador_driverdegradation' = false);
 [SystemFailure] (SystemFailure) -> (1) : (actuador_lossofdriver' = false) &
 (actuador_lossofmotor' = false) & (actuador_mechanismjamming' = false) &
 (actuador_mechanismdegradation' = false) & (actuador_driverdegradation' = false);
endmodule
formula OmissionSpeed_Actuador_Out1 = actuador_lossofdriver| actuador_lossofmotor
       | actuador_mechanismjamming| LowPower_Monitor_Out1 | OmissionSignal_Component3_Out1;
formula WrongPosition_Actuador_Out1 = actuador_mechanismdegradation | actuador_driverdegradation
       | CorruptedSignal_Component3_Out1;
formula CommissionSpeed_Actuador_Out1 =  actuador_driverdegradation | CommissionSignal_Component3_Out1;
```

Fig. 7. Fragment of Prism specification

The next step is using the Prism model-checker to check whether any critical failure condition probability violates the permitted limit. Considering the tabular information of the ACS, our strategy creates probabilistic temporal formulas to check the following failure conditions:

- *Omission of speed at Actuator output port shall be less than 3.10^{-3} per flight;*
- *Commission of speed at Actuator output port shall be less than $3.\ 10^{-3}$ per flight;*
- *Wrong position signal at Actuator output port shall be less than 3.10^{-3} per flight.*

We verify each failure condition using the formula shown in (1), for instance:

$$S <= 10^{-3} [("OmissionSpeed\_Actuator\_Out1")]$$

After checking this formula, where the exact value of the average probability obtained via steady-state analysis for this situation is $2.54e^{-3}$, Prism returns false, indicating that this failure condition was violated. As we have said previously, this strategy can be performed in a hidden way by instructing the Prism model-checker to check each formula automatically using Simulink plug-ins, for example, in such a way that only when a formula is violated this result can be sent back to engineers. Thus the complete quantitative safety analysis can be hidden from the engineers.

So, from such reports, control engineers must adjust the system design by inserting more fault-tolerance features to avoid such failure violations. When all safety requirements are satisfied, the current system design (including its failure and repair rates) is acceptable. To show this analysis to certification authorities, the Markov model can be extracted from Prism by using tools like SHARPE or HARP [20].

Furthermore, one can also investigate scenarios of different phases and maintenance strategies using graphs of the instantaneous probabilities during a certain time interval. For instance, Fig. 8 is the result of evaluating the following formula defined in (2), setting the T parameter from 0 to 100 hours.

P =? [***true*** U<=T ("WrongPosition_Actuador_Out1")]

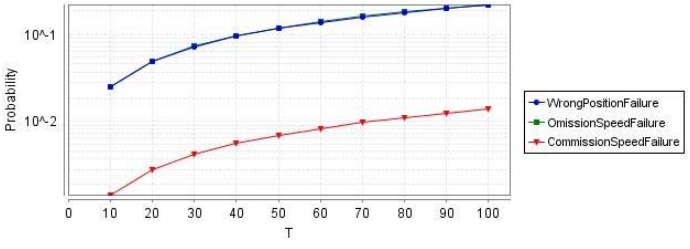


Fig. 8. Instantaneous probability during a period of time

5 Related Work

A large amount of work has been done for quantitative safety assessment which is based mainly on a previous qualitative analysis. An example of an effort in this direction is the use of FTA to compute the failure conditions probability such as the HAZOP [14] which evolves a design developed in Simulink; another relevant effort is the ISAAC project where SCADE is used for modeling and safety analysis [13, 11]. It is also worth mentioning FSAP/NuSMV-SA [15], a fault injection approach developed in the ESACS project. Due the limitations of FTA methods, as discussed in Section 1, more recently approaches considering dynamic reliability have been proposed, based on timed-probabilistic models. We highlight two recent approaches.

In the COMPASS project [22], the model-based probabilistic safety assessment is based on the SLIM (System-Level Integrated Modeling) design language. The approach allows the extension of nominal model of the system adding probabilistic fault behavior, providing a precise characterization of them based on a formal semantics. The analysis is based on a set of verification tools (NuSMV, FSAP, RAT, Sigref, and MRMC) which allows verifying safety/dependability aspects and quantitative analyses (probabilistic analysis of dynamic FTA). The completeness and consistency of this approach qualify it as a promising solution, but the formal modeling language adopted is exposed to the user, demanding that engineers be familiar with this notation. Thus, the impact for the adoption of this solution might be significant; our approach follows the hidden formal methods view.

The work reported in [18] (which proposes pFMEA or Probabilistic FMEA) also uses the Prism model-checker to support quantitative analysis. The approach inte-

grates the failure behavior into the system model described in CTMC via failure injection. In one sense, pFMEA performs a more detailed analysis than ours because it considers faulty as well as normal behaviors of a system. However, this approach does not generate the model systematically, so there is no notion of soundness concerning the model generation, and is more likely to generate state explosion, since it does not present techniques to enable reduction of the Markov model generated.

6 Conclusion

In this paper we propose a systematic strategy to perform quantitative safety assessment of critical systems. Our approach generates a Prism specification from a Simulink diagram, annotated with failure logic. The strategy also creates CSL formulas that check whether all safety requirements are satisfied.

Prism specifications are interesting because they allow the creation and analysis of Markov chains in a more user-friendly and concise way. They also ease the exploration of aspects such as latent and evident failure, monitoring and repair schedule, which are essential to aeronautical systems, for example.

Moreover, if we consider that the traditional fault-tree model is constructed to assess the cause and probability of a single undesirable failure condition, the effort and number of trees generated to perform the analysis of each failure condition are extremely large, making the process expensive [2, 10]. With Markov chains, for instance, those created via Prism, it is possible to represent all failure conditions of a system with a single model. Also, checking the CSL formulas is less expensive than creating fault-trees. Furthermore, engineers can use the Prism specification (Markov chains) to investigate dynamic aspects of a system: experiments to check existing failure scenarios and Phased Mission can be performed by simply changing the values of local variables of the model [2, 8]. However, the current implementation of Prism also imposes some limitations. We cannot generate counter-examples when some property is violated. Fortunately, recent researches are already identifying counter-examples of stationary models, allowing a better traceability of the basic failures and facilitating the cycle of checking and validating the system design [21]. Unfortunately, this solution is not available in Prism yet.

As future work we intend to mechanize the translation strategy and incorporate it as a plug-in in the Matlab/Simulink software. Another improvement to this work is to consider dynamic behavior for failure recovery, capturing the dynamic information in the same way as the static information. In another direction, we intend to prove its soundness and completeness, showing that our strategy creates a Prism specification whose Markov model is equivalent to a Markov semantics given to Simulink diagrams annotated with failure logic and that it can handle any Simulink diagram.

Acknowledgments. This work was partially supported by the National Institute of Software Engineering (INES[3]), funded by CNPq and FACEPE, grants 573964/2008-4 and APQ-1037-1.03/08, by CNPq grant 482462/2009-4 and by the Brazilian Space Agency (UniEspaço 2009).

[3] http://www.ines.org.br

References

1. Serra, P.R.: Safety Assessment of aircraft systems, 2nd edn (2008)
2. ARP 4761: Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems, SAE Inc. (November 1996)
3. Stamatelatos, M., et al.: Fault Tree Handbook with Aerospace Applications. NASA Office of Safety and Mission Assurance, Washington (August 2002)
4. Mota, A., Gomes, A., Jesus, J., Ferri, F., Watanabe, E.: Evolving a Safe System Design Iteratively. Accepted for publication in Proceedings of SAFECOMP (2010)
5. Papadopoulos, Y., McDermid, J., Sasse, R., Heiner, G.: Analysis and synthesis of the behaviour of complex programmable electronic systems in conditions of failure. Reliability Engineering & System Safety 71(3), 229–247 (2001)
6. Alexander, R.D., Kelly, T.P.: Escaping the non-quantitative trap. In: 27th International System Safety Conference, pp. 69–95 (2009)
7. Kwiatkowska, M., Norman, G., Parker, D.: PRISM: Probabilistic Model Checking for Performance and Reliability Analysis. ACM SIGMETRICS Performance Evaluation Review 36(4), 40–45 (2009)
8. Kwiatkowska, M., Norman, G., Parker, D.: Quantitative analysis with the Probabilistic Model Checker PRISM. Electronic Notes in Theoretical Computer Science 153(2), 5–31 (2005)
9. The MathWorks Inc. Simulink User's Guide (2008)
10. McDermid, J.A., Lisagor, O., Pumfrey, D.J.: Towards a Practicable Process for Automated Safety Analysis. In: 24th Int. System Safety Conference, pp. 596–607 (2006)
11. Joshi, A., Heimdahl, M.P.: Model-Based Safety Analysis of Simulink Models Using SCADE Design Verifier. In: Winther, R., Gran, B.A., Dahll, G. (eds.) SAFECOMP 2005. LNCS, vol. 3688, pp. 122–135. Springer, Heidelberg (2005)
12. Software considerations in airborne systems and equipment certification. DO-178B, RTCA Inc., Washington D.C. (December 1992)
13. Kerlund, O.A., et al.: ISAAC, A framework for integrated safety analysis of functional, geometrical and human aspects. In: ERTS (2006)
14. Papadopoulos, Y., Maruhn, M.: Model-based synthesis of fault trees from Matlab-Simulink models. In: Inter. Conference on Dependable Systems and Networks (2001)
15. Bozzano, M., Villafiorita, A.: Improving system reliability via model checking: The FSAP/NuSMV-SA safety analysis platform. In: Anderson, S., Felici, M., Littlewood, B. (eds.) SAFECOMP 2003. LNCS, vol. 2788, pp. 49–62. Springer, Heidelberg (2003)
16. Haverkort, B.R.: Markovian models for performance and dependability evaluation. In: Brinksma, E., Hermanns, H., Katoen, J.-P. (eds.) EEF School 2000 and FMPA 2000. LNCS, vol. 2090, pp. 38–83. Springer, Heidelberg (2001)
17. Saglimbene, M. S.: Reliability analysis techniques: How they relate to aircraft certification. In: Reliability and Maintainability Symposium, pp. 218–222 (2009)
18. Grunske, L., Colvin, R., Winter, K.: pFMEA: Probabilistic Model-Checking Support for FMEA. In: 4th Int. Conference on the QEST (2007)
19. Gomes, A.: Technical Report, `http://www.cin.ufpe.br/~ajog/tr.pdf`
20. Siewiorek, D., Swarz, R.: Reliable Computer System: Design and Evaluation, 3rd edn (1998)
21. Aljazzar, H., et al.: Safety Analysis of an Airbag System Using Probabilistic FMEA and Probabilistic Counterexamples. In: QEST, pp. 299–308 (2009)
22. Bozzano, M., et al.: The COMPASS Approach: Correctness, Modelling and Performability of Aerospace Systems. In: Proceedings of the 28th Int. Conference on Computer Safety, Reliability and Security, September 15-18 (2009)
23. Alejandro, D., et al.: An integrated methodology for the dynamic performance and reliability evaluation of fault-tolerant systems. Reliability Engineering & System Safety 93(11), 1628–1649 (2008)

Learning Techniques for Software Verification and Validation – Special Track at ISoLA 2010

Dimitra Giannakopoulou and Corina S. Păsăreanu

Carnegie Mellon University/NASA Ames Research Center,
Moffett Field, CA 94035, USA
{dimitra.giannakopoulou,corina.s.pasareanu}@nasa.gov

Learning techniques are used increasingly to improve software verification and validation activities. For example, automata learning techniques have been used for extracting behavioral models of software systems. Such models can serve as formal documentation of the software and they can be further verified using automated tools or used for model-based test case generation. Automata learning techniques have also been used for automating compositional verification and for building abstractions of software behavior in the context of symbolic or parameterized model checking. Furthermore, various machine-learning techniques have been used in fine-tuning heuristics used in constraint solving, in coming up with new abstraction techniques in the context of bounded model checking or shape analysis, in inferring invariants of parameterized systems, or in classifying data in black box testing.

This Special Track aims at bringing together researchers and practitioners working on the integration of learning techniques in verification and validation activities for software systems. It particularly emphasizes the use of learning in practice, where it is important to take into account constraints such as performance, scalability, or even limited system availability. The Special Track is part of the 2010 ISoLA International Symposium on Leveraging Applications of Formal Methods, Verification, and Validation.

The Special Track's topics of interest include learning techniques in the context of:

- Synthesis of Behavioral Models for Software Components
- Automated Compositional Verification
- Requirements Elicitation and Analysis
- Abstraction Refinement and Shape Analysis
- Symbolic and Parameterized Verification
- Decision Procedures and Constraint Solving
- Black-box and White-box Testing
- Improving practical applications in avionics, telecommunications, etc.

This year, the Special Track included five presentations, all related to automata learning. Some of the presentations concentrated more on the presentation and evaluation of learning techniques, while others targeted specific application domains such as protocols, and demonstrated the use of learning in the context of such applications. The Track also featured an invited talk related to RERS,

T. Margaria and B. Steffen (Eds.): ISoLA 2010, Part I, LNCS 6415, pp. 640–642, 2010.

a new initiative aiming at the establishment of a community of researchers and practitioners interested in the practical application of automata learning technology. The first RERS meeting was co-located with ISOLA 2010, thus making this year's ISOLA a particularly fertile ground for the discussion of learning-related research ideas.

More specifically, the first presentation, "Learning NFAs", by Martin Leuker, discusses NL*, a learning algorithm for inferring non-deterministic finite-state automata using membership and equivalence queries. The algorithm learns residual finite-state automata (RFSA) in a way similar to Angluins popular L* algorithm, which, however, learns deterministic finite state automata (DFA). Unlike DFAs and NFAs (non-deterministic finite state automata) which characterize the whole class of regular languages, RFSAs are a sub-class of NFAs. RFSAs share important properties with DFAs that make them amenable to learning techniques, i.e. for every regular language, there is a unique minimal canonical RFSA accepting it. Furthermore, RFSAs can be exponentially more succinct than DFAs. They are, therefore, the preferable choice for many learning applications.

The second presentation, "Comparing Learning Algorithms in Automated Assume-Guarantee Reasoning", by Yu-Fang Chen, Edmund M. Clarke, Azadeh Farzan, Fei He, Ming-Hsien Tsai, Yih-Kuen Tsay, Bow-Yaw Wang, and Lei Zhu, is concerned with comparing two learning algorithms for generating contextual assumptions in automated assume-guarantee reasoning. The two algorithms are: CDNF – which implicitly represents contextual assumptions by a conjunction of DNF formulae, and OBDD —which uses ordered binary decision diagrams as its representation. The performance of assume-guarantee reasoning using the two algorithms is compared with monolithic interpolation-based model checking in parameterized hardware test cases.

The third presentation, "Inferring Compact Models of Communication Protocol Entities", by Therese Berg, Bengt Jonsson, Siavash Soleimanifard, uses regular inference (aka automata learning) to generate protocol models. While most existing techniques generate *flat* automata models, the approach presented by the authors is able to generate models enriched with control locations and state variables that describe the dynamic behavior of protocols in a more suitable way. The authors discuss how they applied parts of their approach to an executable state machine specification of the Mobile Arts Advanced Mobile Location Center (A-MLC) protocol and evaluated the results by comparing them to the original specification.

The fourth presentation, "Inference and Abstraction of the Biometric Passport", by Fides Aarts, Julien Schmaltz, Frits Vaandrager, addresses the problem of learning models for model-based testing. To make the approach practical, the authors propose an abstraction technique to reduce the alphabet and the large data sets that are used for learning. Informal documentation and requirements are used to extract apriori knowledge about the teacher and to use this knowledge to define equivalence classes for obtaining a reduced alphabet. The approach is demonstrated by learning a model of the new biometric passport;

the learned model is of comparable size and complexity to a previous model that was developed manually for testing a passport implementation.

Finally, the fifth presentation, "From Zulu to RERS", by Falk Howar, Bernhard Steffen, Maik Merten, summarizes the authors experience with the ZULU challenge on active learning without equivalence queries. This challenge was motivated by the lack of realism of the equivalence queries that are present in many theoretical learning frameworks: indeed, in essentially all practical applications, equivalence queries need to be approximated by the so-called membership queries, which can be implemented using testing. The authors present a winning solution to the challenge and also discuss how their approach can be generalized to a framework for the systematic investigation of domain-specific, scalable learning solutions for applications of practical relevance. In particular, they discuss the RERS initiative, which provides a community platform together with a learning framework that allows users to interactively compose complex learning solutions on the basis of libraries for various learning components, system connectors, and other auxiliary functionality. This framework will be the backbone for an extended challenge on learning in 2011.

Comparing Learning Algorithms in Automated Assume-Guarantee Reasoning*

Yu-Fang Chen[1], Edmund M. Clarke[2], Azadeh Farzan[3], Fei He[4], Ming-Hsien Tsai[5], Yih-Kuen Tsay[5], Bow-Yaw Wang[1,4,6], and Lei Zhu[4]

[1] Academia Sinica, Taiwan
[2] Carnegie Mellon University, USA
[3] University of Toronto, Canada
[4] Tsinghua University, China
[5] National Taiwan University, Taiwan
[6] INRIA, France

Abstract. We compare two learning algorithms for generating contextual assumptions in automated assume-guarantee reasoning. The CDNF algorithm implicitly represents contextual assumptions by a conjunction of DNF formulae, while the OBDD learning algorithm uses ordered binary decision diagrams as its representation. Using these learning algorithms, the performance of assume-guarantee reasoning is compared with monolithic interpolation-based Model Checking in parametrized hardware test cases.

1 Introduction

Suppose one would like to verify whether the composition of M_0 and M_1 satisfies a property π. Consider the following assume-guarantee reasoning rule:

$$\frac{M_0 \| A \models \pi \qquad M_1 \preceq A}{M_0 \| M_1 \models \pi}$$

The rule states that it suffices to find a contextual assumption A such that the composition of M_0 and A satisfies the property, and that M_1 is simulated by A.

* This research was sponsored by the GSRC under contract no. 1041377 (Princeton University), National Science Foundation under contracts no. CCF0429120, no. CNS0926181, no. CCF0541245, and no. CNS0931985, Semiconductor Research Corporation under contract no. 2005TJ1366, General Motors under contract no. GM-CMUCRLNV301, Air Force (Vanderbilt University) under contract no. 18727S3, the Office of Naval Research under award no. N000141010188, the National Science Council of Taiwan projects no. NSC97-2221-E-001-003-MY3, no. NSC97-2221-E-001-006-MY3, no. NSC97-2221-E-002-074-MY3, and no. NSC99-2218-E-001-002-MY3, Natural Sciences and Engineering Research Council of Canada NSERC Discovery Award, Chinese National 973 Plan under grant no. 2010CB328003, the NSF of China under grants no. 60635020, 60903030 and 90718039, the FORMES Project within LIAMA Consortium, and the French ANR project SIVES ANR-08-BLAN-0326-01.

T. Margaria and B. Steffen (Eds.): ISoLA 2010, Part I, LNCS 6415, pp. 643–657, 2010.

If verifying $M_0 \| A \models \pi$ requires less resources than verifying $M_0 \| M_1 \models \pi$, the scalability of verification can be improved by finding such contextual assumptions. Indeed, complete information about M_1 may not be necessary for verifying the property π. Oftentimes, simple contextual assumptions are sufficient to establish properties of interest. Assume-guarantee reasoning offers the flexibility to simplify the verification problem with respect to properties. It is considered as a viable technique to alleviate the state explosion problem.

To effectively apply assume-guarantee reasoning, it is essential to construct a contextual assumption that fulfills the premises and admits efficient verification. By applying the L^* learning algorithm for finite automata [1] and devising a mechanical teacher to answer queries, the learning-based technique in [11] successfully infers contextual assumptions without human intervention. The scalability of automated assume-guarantee reasoning is further improved in [7]. Adopting an implicit representation and applying instead the CDNF learning algorithm for Boolean functions [3], the new technique is able to take advantages of the succinct representation and infer contextual assumptions of larger sizes. Preliminary experimental results show that automated assume-guarantee reasoning through implicit reasoning can outperform the monolithic interpolation-based Model Checking algorithm in some parametrized test cases [7].

Because of their potential applications in practice, several learning algorithms for Boolean functions have been developed [3,12,19]. In this paper, we investigate two of these learning algorithms in the context of automated assume-guarantee reasoning through implicit learning. We compare the performance of the CDNF algorithm [3] and a learning algorithm for ordered binary decision diagrams (OBDD's) [2,12] in automated assume-guarantee reasoning. Both learning algorithms use the same learning model proposed in [1]. The CDNF algorithm is based on the monotone theory and represents an arbitrary Boolean function as a conjunction of DNF formulae [3]. It learns any Boolean function with a polynomial number of queries in the number of variables, and the minimal CNF size and the minimal DNF size of the target function.

The OBDD learning algorithm, on the other hand, is based on the L^* algorithm [12]. For a fixed variable ordering, a valuation on n Boolean variables can be represented by a string in $\{0,1\}^n$. A set of valuations hence corresponds to a finite language. Since any finite language is regular, the L^* algorithm can infer the minimal deterministic finite automaton recognizing the satisfying valuations of any Boolean function. The minimal deterministic finite automaton in turn can be transformed to an OBDD. It is shown that any OBDD is learnable with a polynomial number of queries in the size of the target OBDD.

The two learning algorithms have very different characteristics. Subsequently, their practical costs in the context of assume-guarantee reasoning are not at all clear. To investigate this issue, we assess the effectiveness of both learning algorithms by an extensive set of parametrized test cases. Using two different learning algorithms to infer contextual assumptions, we compare the performance of automated assume-guarantee through implicit learning against the monolithic interpolation-based Model Checking algorithm in [18]. Five parametrized

hardware test cases are taken: the MSI cache coherence protocol [4], the PCI bus protocol [5], a simple bus control protocol [10], the Gigamax cache coherence protocol [9], and synchronous bus arbiters [17]. Each test case has over 15 experiments of different sizes. Three different algorithms are compared in each experiment. Our extensive experiments hopefully can give insights to research directions.

In [15], the CDNF algorithm is used to generate propositional loop invariants in sequential programs. The same learning algorithm is used in [7] to infer contextual assumptions for assume-guarantee reasoning. Applying algorithmic learning to generate contextual assumptions was first proposed in [11]. Following that work, many optimizations have been proposed (see, for example, [20,6,21,13,8]). These optimizations explicitly generate deterministic finite automata as contextual assumptions. In contrast, the work [7] implicitly infers nondeterministic finite automata as contextual assumptions and improves the scalability.

This paper is organized as follows. After preliminaries (Section 2), the learning model and automated assume-guarantee reasoning through implicit learning are reviewed in Section 3 and 4 respectively. They are followed by brief descriptions of the learning algorithms (Section 5). Section 6 gives the experimental results. We conclude in Section 7.

2 Preliminaries

$\mathbb{B} = \{\mathsf{F}, \mathsf{T}\}$ is the Boolean domain. Let $\mathbf{x}$ be a set of Boolean variables and $|\mathbf{x}|$ the size of $\mathbf{x}$. A *Boolean function* $\theta(\mathbf{x})$ *over* $\mathbf{x}$ is a function from $\mathbb{B}^{|\mathbf{x}|}$ to $\mathbb{B}$. We also define $\mathbf{x}'$ to be the set of Boolean variables $\{x' : x \in \mathbf{x}\}$.

A *valuation* $\nu : \mathbf{x} \to \mathbb{B}$ *over* $\mathbf{x}$ is a function from Boolean variables to truth values. Let $\phi(\mathbf{x})$ be a Boolean function over $\mathbf{x}$ and ν a valuation over $\mathbf{x}$. If $\mathbf{y} \subseteq \mathbf{x}$ is a set of Boolean variables, $\nu{\downarrow_{\mathbf{y}}}$ is the *restriction* of ν on $\mathbf{y}$. That is, $\nu{\downarrow_{\mathbf{y}}}$: $\mathbf{y} \to \mathbb{B}$ and $\nu{\downarrow_{\mathbf{y}}}\ (y) = \nu(y)$ for all $y \in \mathbf{y}$. We write $\phi[\nu]$ for the result of evaluating ϕ by replacing each $x \in \mathbf{x}$ with $\nu(x)$. Moreover, let $\psi(\mathbf{x}, \mathbf{x}')$ be a Boolean function over $\mathbf{x}$ and $\mathbf{x}'$. If ν and ν' are valuations over $\mathbf{x}$, we write $\psi[\nu, \nu']$ for the result of evaluating ψ by replacing each $x \in \mathbf{x}$ with $\nu(x)$ and each $x' \in \mathbf{x}'$ with $\nu'(x)$. For example, assume $\nu(x) = \mathsf{F}$ and $\nu'(x) = \mathsf{T}$. If $\phi(x) = \neg x$, $\phi[\nu] = \mathsf{T}$ and $\phi[\nu'] = \mathsf{F}$. If $\psi(x, x') = \neg x \wedge x'$, $\psi[\nu, \nu'] = \mathsf{T}$ and $\psi[\nu', \nu] = \mathsf{F}$.

A *transition system* $M = (\mathbf{x}, \iota(\mathbf{x}), \tau(\mathbf{x}, \mathbf{x}'))$ consists of its *state variables* $\mathbf{x}$, its *initial predicate* $\iota(\mathbf{x})$, and its *transition relation* $\tau(\mathbf{x}, \mathbf{x}')$. A *trace of* M $\alpha = \nu^0\nu^1 \cdots \nu^t$ is a finite sequence of valuations where ν^i is a valuation over $\mathbf{x}$, such that $\iota[\nu^0] = \mathsf{T}$ and $\tau[\nu^i, \nu^{i+1}] = \mathsf{T}$ for $0 \leq i < t$. Define $\mathit{Trace}(M) = \{\alpha : \alpha \text{ is a trace of } M\}$. If $\alpha = \nu^0\nu^1 \cdots \nu^t$ is a finite sequence of valuations over $\mathbf{x}$ and $\mathbf{y} \subseteq \mathbf{x}$, $\alpha{\downarrow_{\mathbf{y}}} = \nu^0{\downarrow_{\mathbf{y}}}\ \nu^1{\downarrow_{\mathbf{y}}} \cdots \nu^t{\downarrow_{\mathbf{y}}}$ is the *restriction* of α on $\mathbf{y}$.

Let $M = (\mathbf{x}, \iota_M(\mathbf{x}), \tau_M(\mathbf{x}, \mathbf{x}'))$ be a transition system. A *state predicate* $\pi(\mathbf{x})$ is a Boolean function over $\mathbf{x}$. We say M *satisfies* π (denoted by $M \models \pi$) if for any $\alpha = \nu^0\nu^1 \cdots \nu^t \in \mathit{Trace}(M)$, we have $\pi[\nu^i] = \mathsf{T}$ for $0 \leq i \leq t$. Given a transition system M and a state predicate π, the *invariant checking* problem is to decide whether M satisfies π. *Model Checking* is an automatic technique to solve the

invariant checking problem. When deciding whether $M \models \pi$, a Model Checking algorithm returns a witness if M does not satisfy π. A *witness* to $M \not\models \pi$ is a trace $\nu^0\nu^1 \cdots \nu^t$ of M such that $\pi(\nu^i) = \mathsf{T}$ for $0 \leq i < t$ but $\pi(\nu^t) = \mathsf{F}$.

Let $N = (\mathbf{x}, \iota_N(\mathbf{x}), \tau_N(\mathbf{x}, \mathbf{x}'))$ be a transition system. We say M *is simulated by* N or N *simulates* M (denoted by $M \preceq N$) if $\forall \mathbf{x}.\iota_M(\mathbf{x}) \implies \iota_N(\mathbf{x})$ and $\forall \mathbf{x}\mathbf{x}'.\tau_M(\mathbf{x}, \mathbf{x}') \implies \tau_N(\mathbf{x}, \mathbf{x}')$ hold. In words, M is simulated by N if the initial condition of M is stronger than that of N and every transition allowed in M is also allowed in N. Clearly, if $M \preceq N$, then $\mathit{Trace}(M) \subseteq \mathit{Trace}(N)$.

Let $\mathbf{x}_i$ be sets of Boolean variables for $i = 0, 1$ ($\mathbf{x}_i$'s are not necessarily disjoint). Consider $M_i = (\mathbf{x}_i, \iota_i(\mathbf{x}_i), \tau_i(\mathbf{x}_i, \mathbf{x}_i'))$ for $i = 0, 1$. The *composition* of M_0 and M_1 is the transition system $M_0 \| M_1 = (\mathbf{x}_0 \cup \mathbf{x}_1, \iota_0(\mathbf{x}_0) \wedge \iota_1(\mathbf{x}_1), \tau_0(\mathbf{x}_0, \mathbf{x}_0') \wedge \tau_1(\mathbf{x}_1, \mathbf{x}_1'))$. Note that for any finite sequence of valuations α over $\mathbf{x}_0 \cup \mathbf{x}_1$, $\alpha \in \mathit{Trace}(M_0 \| M_1)$ if and only if $\alpha{\downarrow_{\mathbf{x}_0}} \in \mathit{Trace}(M_0)$ and $\alpha{\downarrow_{\mathbf{x}_1}} \in \mathit{Trace}(M_1)$.

An *assume-guarantee reasoning rule* is of the form $\dfrac{\Theta_0 \cdots \Theta_m}{\Delta}$ where $\Theta_0, \ldots, \Theta_m$ are its *premises* and Δ its *conclusion*. An assume-guarantee reasoning rule is *sound* if its conclusion holds when its premises are fulfilled. A rule is *invertible* if its premises can be fulfilled when its conclusion holds. We use the following assume-guarantee reasoning rule throughout the paper:

Lemma 1. *Let $M_i = (\mathbf{x}_i, \iota_i(\mathbf{x}_i), \tau_i(\mathbf{x}_i, \mathbf{x}_i'))$ be transition systems for $i = 0, 1$, and π a state predicate over $\mathbf{x}_0 \cup \mathbf{x}_1$. The following rule is sound and invertible:*

$$\frac{M_0 \| A \models \pi \qquad M_1 \preceq A}{M_0 \| M_1 \models \pi} \tag{1}$$

where $A = (\mathbf{x}_1, \iota_A(\mathbf{x}_1), \tau_A(\mathbf{x}_1, \mathbf{x}_1'))$ is a transition system.

Let $M_i = (\mathbf{x}_i, \iota_i(\mathbf{x}_i), \tau_i(\mathbf{x}_i, \mathbf{x}_i'))$ be transition systems for $i = 0, 1$ and π a state predicate over $\mathbf{x}_0 \cup \mathbf{x}_1$, a transition system $A = (\mathbf{x}_1, \iota_A(\mathbf{x}_1), \tau_A(\mathbf{x}_1, \mathbf{x}_1'))$ such that $M_0 \| A \models \pi$ and $M_1 \preceq A$ is called a *contextual assumption* of M_0.

3 The Learning Model

Before reviewing the learning-based approach to inferring contextual assumptions, we briefly describe the learning model used in [1,3,7]. For any unknown *target* Boolean function $\lambda(\mathbf{x})$ over a fixed set of Boolean variables $\mathbf{x}$, an exact learning algorithm for Boolean functions computes a representation of $\lambda(\mathbf{x})$ by interacting with a teacher. The *teacher* knows the Boolean function $\lambda(\mathbf{x})$ and answers two types of queries made by the learning algorithm:

- *Membership query $\mathit{MEM}(\nu)$ for the target $\lambda(\mathbf{x})$,* where ν is a valuation over $\mathbf{x}$. If $\lambda[\nu] = \mathsf{T}$, the teacher answers *YES*; and *NO*, otherwise.
- *Equivalence query $\mathit{EQ}(\theta)$ for the target $\lambda(\mathbf{x})$,* where $\theta(\mathbf{x})$ is a Boolean function over $\mathbf{x}$. If the *conjecture* $\theta(\mathbf{x})$ is equivalent to the target Boolean function $\lambda(\mathbf{x})$, the teacher answers *YES*. Otherwise, the teacher provides a valuation ν over $\mathbf{x}$ where $\theta[\nu] \neq \lambda[\nu]$. The valuation ν serves as a *counterexample* to the equivalence query $\mathit{EQ}(\theta)$.

Assume $\lambda(x, y) = (x \wedge \neg y) \vee (\neg x \wedge y)$ is the target Boolean function over x and y. If the learning algorithm makes the query $MEM(\nu_0)$ where $\nu_0(x) = \nu_0(y) = \mathsf{F}$ (denoted by $\nu_0(xy) = \mathsf{FF}$), the teacher answers NO for $\lambda(\mathsf{F}, \mathsf{F}) = \mathsf{F}$. For a different valuation $\nu_1(xy) = \mathsf{TF}$, the teacher answers YES. Similarly, consider the equivalence query $EQ(x \vee y)$. The teacher should provide the valuation $\nu_2(xy) = \mathsf{TT}$ as a counterexample, since $\mathsf{T} \vee \mathsf{T} = \mathsf{T} \neq \mathsf{F} = \lambda(\mathsf{T}, \mathsf{T})$. For another equivalence query $EQ((x \vee \neg y) \wedge (\neg x \vee y))$, the teacher answers YES.

4 Learning a Contextual Assumption

In automated assume-guarantee reasoning through learning, one applies an exact learning algorithm to infer a contextual assumption that fulfills both premises of the assume-guarantee reasoning rule (1). In order to do so, a mechanical teacher is designed to answer queries from the learning algorithm. Assume $A = (\mathbf{x}_1, \iota_A(\mathbf{x}_1), \tau_A(\mathbf{x}_1, \mathbf{x}'_1))$ is a contextual assumption satisfying both premises. The teacher is required to resolve four types of queries:

- the membership query $MEM(\mu)$ for the target $\iota_A(\mathbf{x}_1)$;
- the membership query $MEM(\mu, \mu')$ for the target $\tau_A(\mathbf{x}_1, \mathbf{x}'_1)$;
- the equivalence query $EQ(\iota)$ for the target $\iota_A(\mathbf{x}_1)$; and
- the equivalence query $EQ(\tau)$ for the target $\tau_A(\mathbf{x}_1, \mathbf{x}'_1)$.

If A was known, it would be straightforward to design such a mechanical teacher. All queries can easily be resolved by evaluating or comparing the initial predicate or the transition relation of the purported contextual assumption. However, such a contextual assumption is yet to be inferred and current unknown to us. We thus look for a replacement in the design of the mechanical teacher.

In [7], the mechanical teacher simply uses M_1 in place of the unknown contextual assumption. Clearly, inferring M_1 as the contextual assumption in the assume-guarantee reasoning rule (1) is not beneficial: the first premise is precisely the conclusion when the contextual assumption A is M_1. However, several conjectures will be proposed while the learning algorithm is inferring M_1. If one of them satisfies both premises, it can serve as a contextual assumption and conclude the verification. Since contextual assumptions are not unique, one expects that another contextual assumption will be generated before M_1 is inferred.

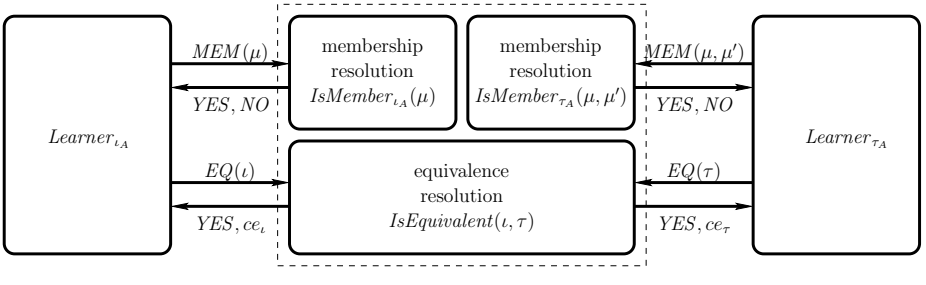


Fig. 1. Structure of Contextual Assumption Generator

We adopt the architecture of the mechanical teacher proposed in [7]. Recall that a contextual assumption $A = (\mathbf{x}_1, \iota_A(\mathbf{x}_1), \tau_A(\mathbf{x}_1, \mathbf{x}'_1))$ consists of two Boolean formulae. We hence deploy two instances of the learning algorithm (Figure 1): one infers the initial predicate $\iota_A(\mathbf{x}_1)$; the other infers the transition relation $\tau_A(\mathbf{x}_1, \mathbf{x}'_1)$. In the figure, the instances of the learning algorithm are shown on the sides. The instance $Learner_{\iota_A}$ is intended to compute the initial predicate $\iota_A(\mathbf{x}_1)$; the instance $Learner_{\tau_A}$ is to compute the transition relation $\tau_A(\mathbf{x}_1, \mathbf{x}'_1)$. The mechanical teacher is denoted by the dashed box in the middle.

The teacher consists of three query resolution algorithms. The algorithms $IsMember_{\iota_A}(\mu)$ and $IsMember_{\tau_A}(\mu, \mu')$ resolve membership queries for the initial predicate and the transition relation respectively (Algorithm 1). Since the mechanical teacher uses M_1 as the target, membership queries are resolved by evaluating the initial predicate or the transition relation of M_1 respectively.

Input: $MEM(\mu)$: a membership query for the target $\iota_A(\mathbf{x}_1)$
Output: *YES* or *NO*
`/* ` $\iota_1(\mathbf{x}_1)$ `is the initial predicate of` M_1 `*/`
if $\iota_1[\mu] = \mathsf{T}$ **then return** *YES* **else return** *NO*;

(a) $IsMember_{\iota_A}(\mu)$

Input: $MEM(\mu, \mu')$: a membership query for the target $\tau_A(\mathbf{x}_1, \mathbf{x}'_1)$
Output: *YES* or *NO*
`/* ` $\tau_1(\mathbf{x}_1, \mathbf{x}'_1)$ `is the transition relation of` M_1 `*/`
if $\tau_1[\mu, \mu'] = \mathsf{T}$ **then return** *YES* **else return** *NO*;

(b) $IsMember_{\tau_A}(\mu, \mu')$

Algorithm 1. Membership Query Resolution Algorithms

The algorithm $IsEquivalent(\iota, \tau)$ resolves both types of equivalence queries (Algorithm 2). The equivalence query resolution algorithm waits until the equivalence query $EQ(\iota)$ from $Learner_{\iota_A}$ and the equivalence query $EQ(\tau)$ from $Learner_{\tau_A}$ are available. It then checks the premise $M_1 \preceq C$ in the assume-guarantee reasoning rule (11) with $C = (\mathbf{x}_1, \iota(\mathbf{x}_1), \tau(\mathbf{x}_1, \mathbf{x}'_1))$. If $M_1 \not\preceq C$, a counterexample is returned to either $Learner_{\iota_A}$ or $Learner_{\tau_A}$ to refine the current conjecture. For instance, assume $\forall \mathbf{x}_1.\iota_1(\mathbf{x}_1) \Rightarrow \iota(\mathbf{x}_1)$ is false. There is a valuation μ such that $\iota_1[\mu] = \mathsf{T}$ and $\iota[\mu] = \mathsf{F}$. The equivalence query resolution algorithm returns μ to $Learner_{\iota_A}$ as the counterexample to the equivalence query $EQ(\iota)$. It then waits for another equivalence query $EQ(\iota')$ from $Learner_{\iota_A}$, and restarts with the new conjectured transition system $C' = (\mathbf{x}_1, \iota'(\mathbf{x}_1), \tau(\mathbf{x}_1, \mathbf{x}'_1))$.

Assume the equivalence query resolution algorithm has verified $M_1 \preceq C$. It then checks if the other premise $M_0 \| C \models \pi$ is fulfilled. If so, we have found a contextual assumption that establishes the property. Otherwise, there is a trace α witnessing $M_0 \| C \not\models \pi$. The equivalence query resolution algorithm then invokes $IsWitness(\alpha)$ to analyze the trace α.

Input: $EQ(\iota)$: an equivalence query for the target $\iota_A(\mathbf{x}_1)$; $EQ(\tau)$: an equivalence query for the target $\tau_A(\mathbf{x}_1, \mathbf{x}'_1)$
Output: *YES*, a counterexample to $EQ(\iota)$, or a counterexample to $EQ(\tau)$
let C be the transition system $(\mathbf{x}_1, \iota(\mathbf{x}_1), \tau(\mathbf{x}_1, \mathbf{x}'_1))$;
if $\iota_1(\mathbf{x}_1) \wedge \neg\iota(\mathbf{x}_1)$ *is satisfied by* μ **then**
  answer $EQ(\iota)$ with the counterexample μ;
  receive another equivalence query $EQ(\iota')$;
  call *IsEquivalent*(ι', τ);
if $\tau_1(\mathbf{x}_1, \mathbf{x}'_1) \wedge \neg\tau(\mathbf{x}_1, \mathbf{x}'_1)$ *is satisfied by* $\mu\mu'$ **then**
  answer $EQ(\tau)$ with the counterexample $\mu\mu'$;
  receive another equivalence query $EQ(\tau')$;
  call *IsEquivalent*(ι, τ');
if $M_0 \| C \models \pi$ **then**
  answer $EQ(\iota)$ with *YES*;
  answer $EQ(\tau)$ with *YES*;
  report "$M_0 \| M_1 \models \pi$";
else
  let α be a witness to $M_0 \| C \not\models \pi$;
  call *IsWitness*(α);
end

Algorithm 2. *IsEquivalent*(ι, τ)

The witness analysis algorithm *IsWitness*(α) checks if the restriction $\alpha{\downarrow_{\mathbf{x}_1}}$ is also a trace of M_1 (Algorithm 3). If so, α is in fact a witness to $M_0 \| M_1 \not\models \pi$. Otherwise, the restriction $\alpha{\downarrow_{\mathbf{x}_1}}$ must deviate from the initial predicate or the transition relation of M_1. The witness analysis algorithm therefore returns the deviation as a counterexample to either $EQ(\iota)$ or $EQ(\tau)$. It then waits for a new equivalence query and restarts the equivalence query resolution algorithm.

The correctness of the algorithm is established by the following properties [7].

Lemma 2 (soundness). *Let* $M_i = (\mathbf{x}_i, \iota_i(\mathbf{x}_i), \tau_i(\mathbf{x}_i, \mathbf{x}'_i))$ *be transition systems for* $i = 0, 1$, *and* $\pi(\mathbf{x})$ *a state predicate over* $\mathbf{x} = \mathbf{x}_0 \cup \mathbf{x}_1$.

1. *Let* $\iota(\mathbf{x}_1)$ *and* $\tau(\mathbf{x}_1, \mathbf{x}'_1)$ *be Boolean functions over* $\mathbf{x}_1$ *and* $\mathbf{x}_1 \cup \mathbf{x}'_1$ *respectively. If IsEquivalent*(ι, τ) *reports "*$M_0 \| M_1 \models \pi$*," then* $M_0 \| M_1 \models \pi$*;*
2. *Let* $\iota(\mathbf{x}_1)$ *and* $\tau(\mathbf{x}_1, \mathbf{x}'_1)$ *be Boolean functions over* $\mathbf{x}_1$ *and* $\mathbf{x}_1 \cup \mathbf{x}'_1$ *respectively. If IsEquivalent*(ι, τ) *reports "*$M_0 \| M_1 \not\models \pi$ *is witnessed by* α*," then* α *is a witness to* $M_0 \| M_1 \not\models \pi$.

Lemma 3 (completeness). *Let* $M_i = (\mathbf{x}_i, \iota_i(\mathbf{x}_i), \tau_i(\mathbf{x}_i, \mathbf{x}'_i))$ *be transition systems for* $i = 0, 1$, *and* $\pi(\mathbf{x})$ *a state predicate over* $\mathbf{x} = \mathbf{x}_0 \cup \mathbf{x}_1$.

1. *If* $M_0 \| M_1 \models \pi$, *then IsEquivalent*(ι, τ) *reports "*$M_0 \| M_1 \models \pi$*" for some Boolean functions* $\iota(\mathbf{x}_1)$ *and* $\tau(\mathbf{x}_1, \mathbf{x}'_1)$ *over* $\mathbf{x}_1$ *and* $\mathbf{x}_1 \cup \mathbf{x}'_1$ *respectively.*
2. *If* α *is a witness to* $M_0 \| M_1 \not\models \pi$, *then IsEquivalent*$(\iota, \tau)$ *reports "*$M_0 \| M_1 \not\models \pi$ *is witnessed by* α*" for some Boolean functions* $\iota(\mathbf{x}_1)$ *and* $\tau(\mathbf{x}_1, \mathbf{x}'_1)$ *over* $\mathbf{x}_1$ *and* $\mathbf{x}_1 \cup \mathbf{x}'_1$ *respectively.*

Input: α is a witness to $M_0 \| C \not\models \pi$
Output: a counterexample to $EQ(\iota)$, or a counterexample to $EQ(\tau)$
let $\alpha{\downarrow_{\mathbf{x}_1}} = \mu^0\mu^1 \cdots \mu^t$;
if $\iota_1[\mu^0] = \mathsf{F}$ **then**
  answer $EQ(\iota)$ with the counterexample μ^0;
  receive another equivalence query $EQ(\iota')$;
  call *IsEquivalent*(ι', τ);
for $i := 1$ **to** t **do**
  if $\tau_1[\mu^{i-1}, \mu^i] = \mathsf{F}$ **then**
    answer $EQ(\tau)$ with the counterexample $\mu^{i-1}\mu^i$;
    receive another equivalence query $EQ(\tau')$;
    call *IsEquivalent*(ι, τ');
end
report "$M_0 \| M_1 \not\models \pi$ is witnessed by α";

Algorithm 3. *IsWitness*(α)

Lemma 4 (termination). *Let* $M_i = (\mathbf{x}_i, \iota_i(\mathbf{x}_i), \tau_i(\mathbf{x}_i, \mathbf{x}_i'))$ *be transition systems for* $i = 0, 1$, *and* $\pi(\mathbf{x})$ *a state predicate over* $\mathbf{x} = \mathbf{x}_0 \cup \mathbf{x}_1$. *Suppose the learning algorithm infers an unknown target Boolean formula* f *with* $t(|f|)$ *queries. The mechanical teacher reports "*$M_0 \| M_1 \models \pi$*" or "*$M_0 \| M_1 \not\models \pi$ *is witnessed by* α*" with at most* $t(|\iota_1|) + t(|\tau_1|)$ *queries.*

5 Exact Learning Algorithms for Boolean Functions

Thanks to the interface defined in the learning model, the mechanical teacher presented in Section 4 is independent of the underlying learning algorithm. As long as a learning algorithm uses the same learning model, it can be instantiated as $Learner_{\iota_A}$ and $Learner_{\tau_A}$ to infer contextual assumptions. There are in fact several learning algorithms for Boolean functions [3,12,19]. Here, we are most interested in the CDNF algorithm [3] and the OBDD learning algorithm [12].

5.1 The CDNF Algorithm

The CDNF algorithm is an exact learning algorithm for Boolean functions [3]. It represents any unknown target Boolean function in the conjunctive-disjunctive normal form.[1] The CDNF algorithm works iteratively. In each iteration, it first uses membership queries to construct a conjecture in CDNF. After a conjecture is built, the CDNF algorithm poses an equivalence query to check if it has inferred the unknown target. If not, the current conjecture is refined by the counterexample. Depending on whether the conjecture need be weakened or strengthened, the learning algorithm either modifies DNF formulae of the current conjecture, or adds a DNF formula to the conjecture respectively.

Initially, the CDNF algorithm starts with the degenerated conjecture T, and makes the equivalence query $EQ(\mathsf{T})$. If T is not the target, the CDNF algorithm

[1] A Boolean formula is in the *conjunctive-disjunctive normal form (CDNF)* if it is a conjunction of DNF formulae.

adds the first conjunct and therefore strengthens the initial conjecture. More generally, let us say that a counterexample of an equivalence query is *positive* if the target evaluates to T under the counterexample; it is *negative* if the target evaluates to F. When the CDNF algorithm obtains a positive counterexample, it weakens conjuncts of the current conjecture by adding a conjunctive clause to DNF formulae in the conjecture. When the CDNF algorithm obtains a negative counterexample, it strengthens the current conjecture by adding a DNF formula to the conjecture. The first equivalence query simply initiates the iterations by strengthening the degenerated conjunction.

Let $\lambda(\mathbf{x})$ be a Boolean function over $\mathbf{x}$, $|\lambda(\mathbf{x})|_{DNF}$ and $|\lambda(\mathbf{x})|_{CNF}$ denote the sizes of $\lambda(\mathbf{x})$ in minimal disjunctive and conjunctive normal forms respectively. Under the learning model in Section 3, the CDNF algorithm computes a representation for any target Boolean function $\lambda(\mathbf{x})$ with a polynomial number of queries in $|\lambda(\mathbf{x})|_{DNF}$, $|\lambda(\mathbf{x})|_{CNF}$, and $|\mathbf{x}|$ [3].

5.2 A Learning Algorithm for Ordered Binary Decision Diagrams

Fix a variable ordering on Boolean variables $\mathbf{x}$. A valuation over $\mathbf{x}$ can be represented by a string in $\{0,1\}^{|\mathbf{x}|}$. For any Boolean function $\lambda(\mathbf{x})$, its satisfying valuations hence correspond to a finite language. Moreover, an OBDD for $\lambda(\mathbf{x})$ can be seen as a recognizer for the finite language of satisfying valuations. Observe that the structure of the OBDD for $\lambda(\mathbf{x})$ is in fact similar to the minimal deterministic finite automaton for the language of satisfying valuations [16]. Subsequently, one may infer an unknown OBDD by the L^* algorithm. This idea has been explored in an exact learning algorithm for OBDD [12]. Under the learning model in Section 3, the OBDD learning algorithm computes any target OBDD d with a polynomial number of queries in the size of d. By Shannon's expansion, we obtain another exact learning algorithm for Boolean functions.

The OBDD learning algorithm behaves very differently from the CDNF algorithm. As described above, the CDNF algorithm starts with the equivalence query $EQ(\mathsf{T})$. On the other hand, the OBDD learning algorithm almost always starts with the equivalence query $EQ(\mathsf{F})$. Starting from the empty valuation, the L^* algorithm builds its first conjecture by making membership queries on extensions of the empty valuation. If all valuations of length less than two are rejected, the L^* algorithm will build the minimal finite automaton recognizing the empty language as its conjecture. Recall that any satisfying valuation for Boolean functions over n variables must have length n, and that the empty set of satisfying valuations corresponds to the Boolean function F. The OBDD learning algorithm subsequently always starts with the equivalence query $EQ(\mathsf{F})$ when there is more than one Boolean variable.

In our settings, starting with the equivalence query $EQ(\mathsf{F})$ may impede the performance. After $Learner_{\iota_A}$ and $Learner_{\tau_A}$ make their first equivalence queries $EQ(\mathsf{F})$, the equivalence query resolution algorithm has the transition system $C_\perp = (\mathbf{x}_1, \mathsf{F}, \mathsf{F})$. Since $M_1 \not\preceq C_\perp$, it will ask the learning algorithm to weaken both conjectures. In fact, the OBDD learning algorithm sometimes weakens too conservatively and infers M_1 as the contextual assumption.

Input: $MEM(\mu)$: a membership query for the target $\lambda(\mathbf{x})$
Output: *YES* or *NO*
if *teacher's answer to* $MEM(\mu)$ *is YES* **then return** *NO* **else return** *YES*

(a) Inverted Membership Query

Input: $EQ(\theta)$: a membership query for the target $\lambda(\mathbf{x})$
Output: *YES* or a counterexample to $EQ(\theta)$
return *teacher's answer to* $EQ(\neg\theta)$

(b) Inverted Equivalence Query

Algorithm 4. Inverted Queries

One simple way to avoid this problem is to invert queries from the OBDD learning algorithm (Algorithm 4). The main idea is to let the learning algorithm infer the negation of the target Boolean function. When the OBDD learning algorithm makes a membership query, we return *NO* if the teacher answers *YES*. Otherwise, we return *YES*. When the OBDD learning algorithm makes an equivalence query, we ask the teacher if the negation of the conjecture is the target. If not, we forward teacher's counterexample to the learning algorithm. With this simple translation, the first equivalence query $EQ(\mathsf{F})$ from the OBDD learning algorithm is converted to the equivalence query $EQ(\mathsf{T})$ as desired.

6 Experiments

We have implemented a prototype of the mechanical teacher in OCaml. Our implementation uses the OCaml thread library. Each instance of the learning algorithm is executed in a separate thread, and the equivalence query resolution algorithm is executed in a third thread. MiniSat 2 (version 070721) is used to evaluate Boolean functions in Algorithm 1, and check the simulation relation in Algorithm 2. For monolithic Model Checking, we implement the interpolation-based algorithm in [18]. Interpolants are computed by instrumenting MiniSat 2. The interpolation-based Model Checking algorithm is also used in the equivalence query resolution algorithm (Algorithm 2).

We report the following five test cases: the MSI cache coherence protocol [5], the PCI bus protocol [4], a simple bus control protocol [10], the Gigamax cache coherence protocol [9], and synchronous bus arbiters [17] . Each test case has experiments parametrized by the number of nodes. Let $M_1, \ldots, M_n$ be nodes and π a state predicate. We verify $M_1 \| \cdots \| M_n \models \pi$ in an experiment with n nodes. An experiment with n nodes is divided into different partitions in n trials. We apply the following assume-guarantee reasoning rule in the i-th trial:

$$\frac{(M_1 \| \cdots \| M_{i-1} \| M_{i+1} \| \cdots M_n) \| A \models \pi \qquad M_i \preceq A}{(M_1 \| \cdots \| M_{i-1} \| M_{i+1} \| \cdots M_n) \| M_i \models \pi}$$

In each trial, we use the CDNF algorithm and the OBDD learning algorithm to generate a contextual assumption A to verify $M_1 \| \cdots \| M_n \models \pi$. We choose the best result among the n trials and compare it with monolithic Model Checking. All experimental results are collected on a server with 8 Intel Xeon 2.0GHz processors. Each experiment is carried out on a dedicated core with 4GB memory.

MSI Cache Coherence Protocol. In the MSI cache coherence protocol, a memory is shared among n nodes [5]. Each node has a cache. A bus connects the memory and caches of the nodes. When a node accesses a memory cell, it reads the cell from the bus and keeps a copy in its cache. Several copies of the same memory cell can be kept in different nodes. The MSI protocol ensures data coherence by keeping each cache in one of the three states: Modified, Shared, and Invalid [14]. Two properties are verified in the experiments with 4 to 20 nodes (Figure 2). The property `master1` specifies that at most one node can be the bus master. The other property `m0s1m1` states that if the data at a memory location is modified by a node, it cannot be shared or modified by another node at the same time.

nodes	4	5	6	7	8	9	10	11	12
monolithic	3s	5s	9s	13s	15s	33s	44s	1m41s	1m48s
cdnf	**0s**	**0s**	**2s**	**1s**	**3s**	**6s**	**4s**	**7s**	**7s**
bdd	**0s**	**0s**	**2s**	2s	**3s**	**6s**	5s	**7s**	8s

nodes	13	14	15	16	17	18	19	20	avg
monolithic	54s	1m30s	1m51s	50s	3m54s	5m38s	5m32s	5m16s	1m49s
cdnf	**11s**	**18s**	**7s**	**16s**	**48s**	**28s**	**12s**	1m12s	**14s**
bdd	**11s**	**18s**	8s	**16s**	56s	**28s**	14s	**1m10s**	15s

(a) Results for the Property `master1`

nodes	4	5	6	7	8	9	10	11	12
monolithic	**55s**	5m54s	46s	4m21s	22m19s	2m7s	2m37s	2m29s	9m49s
cdnf	2m14s	48s	54s	1m38s	1m18s	1m17s	2m43s	2m35s	2m43s
bdd	2m10s	**22s**	**44s**	**1m35s**	**1m14s**	**1m14s**	**2m41s**	**46s**	**2m38s**

nodes	13	14	15	16	17	18	19	20	avg
monolithic	6m15s	2m44s	11m35s	4m7s	18m55s	9m33s	10m12s	9m31s	7m18s
cdnf	2m17s	2m9s	3m4s	2m44s	2m51s	2m35s	3m45s	4m36s	2m22s
bdd	**2m14s**	**2m8s**	**3m2s**	**2m29s**	**2m50s**	**2m32s**	**3m26s**	**4m13s**	**2m8s**

(b) Results for the Property `m0s1m1`

Fig. 2. Experimental Results for the MSI Protocol

In both figures, we show the verification time of the monolithic interpolation-based Model Checking (*monolithic*), the verification time of assume-guarantee reasoning using the CDNF algorithm (*cdnf*), and those using the OBDD learning algorithm (*bdd*). We also identify the best algorithm in each experiment.

For both properties, assume-guarantee reasoning outperforms monolithic Model Checking consistently. Between the two learning algorithms used in assume-guarantee reasoning, their performances are almost indistinguishable for the property `master1`. The OBDD learning algorithm wins the CDNF algorithm in all experiments in the property `m0s1m1`. In the experiment with 11 nodes, the contextual assumption generated by the OBDD learning algorithm concludes the verification in less a minutes. The CDNF algorithm, on the other hand, takes more than two and a half minutes to verify the same property. It is in fact slower than monolithic Model Checking in this experiment.

PCI. This example models the PCI bus protocol with two levels of arbiters controlling data transmission [4]. For $2n$ PCI devices, we create a first-level arbiter and n second-level arbiters. The first-level arbiter connects all second-level arbiters. Each second-level arbiter connects two devices. When a device wants to start a transaction, it first requests the permission from its second-level arbiter. The second-level arbiter then selects a request between its devices. The first-level arbiter in turn grants the permission to the selected request from one of the second-level arbiters. We check that the first two nodes do not consider the bus to be idle at the same time. Figure 3 gives the experimental results. Assume-guarantee reasoning significantly outperforms monolithic Model Checking in this case. The difference between both learning algorithms is however negligible except for the experiment with 17 nodes. The CDNF algorithm proves the property in 37 seconds and wins the OBDD learning algorithm by 8 seconds. Monolithic Model Checking is unable to conclude the verification in two and a half minutes.

nodes	4	5	6	7	8	9	10	11	12
monolithic	15s	15s	25s	34s	46s	49s	1m2s	1m20s	1m17s
cdnf	**6s**	**10s**	**11s**	**14s**	**16s**	**17s**	**22s**	**24s**	28s
bdd	**6s**	**10s**	**11s**	**14s**	**16s**	**17s**	**22s**	27s	**24s**

nodes	13	14	15	16	17	18	19	20	avg
monolithic	1m25s	1m40s	1m48s	1m54s	2m45s	2m24s	2m34s	3m17s	1m26s
cdnf	**30s**	**33s**	**27s**	30s	**37s**	**40s**	**54s**	**50s**	**26s**
bdd	31s	34s	28s	**29s**	45s	41s	55s	**50s**	27s

Fig. 3. Experimental Results for PCI

Bus Control Protocol. In this bus control protocol, several nodes are attached to the bus [10]. Each node is assigned to a unique priority. A counter is used to decide the ownership of the bus. The node with priority p can send data when the counter has value p. If a node sends data, the counter is reset. Otherwise, the counter is incremented by one. We check that the first node cannot send data on the bus together with any other node at the same time. Figure 4 gives the results for the experiments with 34 to 50 nodes. Assume-guarantee reasoning clearly outperforms monolithic Model Checking. The CDNF algorithm and the OBDD learning algorithm win 10 and 5 of the experiments respectively. They are tied at the first place for the remaining 2 experiments.

nodes	34	35	36	37	38	39	40	41	42
monolithic	32s	33s	32s	33s	49s	2m25s	41s	40s	1m28s
cdnf	**21s**	**18s**	**21s**	**22s**	**21s**	**25s**	34s	**28s**	32s
bdd	**21s**	22s	25s	24s	25s	26s	**28s**	31s	**25s**

nodes	43	44	45	46	47	48	49	50	avg
monolithic	52s	52s	1m0s	54s	54s	1m12s	47s	15m42s	2m33s
cdnf	34s	**31s**	**33s**	**38s**	38s	**39s**	50s	**28s**	**43s**
bdd	**31s**	36s	**33s**	43s	**26s**	40s	**42s**	42s	44s

Fig. 4. Experimental Results for the Bus Control Protocol

nodes	21	22	23	24	25	26	27	28	29
monolithic (sec)	4.827	5.471	5.885	6.236	7.030	8.020	8.380	9.562	10.329
cdnf (sec)	**4.656**	**5.150**	**5.517**	**6.099**	6.789	**7.064**	8.213	8.958	**9.838**
bdd (sec)	4.857	5.328	5.741	6.339	**6.696**	7.693	**8.085**	**8.814**	10.086

nodes	30	31	32	33	34	35	36	37	avg
monolithic (sec)	11.503	12.107	13.175	14.145	14.890	15.791	17.009	18.518	14.524
cdnf (sec)	**10.803**	**11.579**	**12.904**	**13.977**	**14.689**	**15.658**	**16.962**	**17.966**	**14.065**
bdd (sec)	11.399	11.946	13.010	14.003	15.267	16.091	17.256	18.284	14.324

Fig. 5. Experimental Results for the Gigamax Cache Coherence Protocol

Gigamax. In the Gigamax cache coherence protocol, several processors and a memory is attached to a bus [9]. Each processor has a local cache. A local cache can be in the Invalid, Shared, or the Owned state. Among the memory and processors, at most one can be the bus master and issues commands on the bus. When a processor is the bus master and issues a ReadOwned command, its local cache state becomes Owned. A processor can write to the bus if its local cache state is Owned. For this test case, we consider the experiments with 21 to 37 nodes and verify that the memory is written by at most one processor. The results are shown in Figure 5. The CDNF algorithm outperforms the other two in 14 experiments whereas the OBDD learning algorithm wins the remaining 3 experiments with a small margin. The OBDD learning algorithm performs just slightly better than monolithic Model Checking on average.

Synchronous Bus Arbiters. The synchronous bus arbiter is a bus arbitration protocol for synchronous circuits [17]. In this protocol, n nodes are connected in a ring. A token is passed around the nodes. A node can request and acknowledge the token from the node next to it. The node having the token has the exclusive right to access the bus. For this test case, we check that there is at most one node can access to the bus. Figure 6 shows the results. Assume-guarantee reasoning and monolithic Model Checking perform almost identically on smaller experiments. For experiments with more than 12 nodes, assume-guarantee reasoning is slightly better. Monolithic Model Checking however is able to finish the experiment with 20 nodes by 40 seconds. Subsequently, monolithic Model Checking is comparable to assume-guarantee reasoning on average.

nodes	4	5	6	7	8	9	10	11	12
monolithic	**0s**	**1s**	**3s**	**5s**	**10s**	**18s**	33s	**54s**	1m38s
cdnf	**0s**	**1s**	**3s**	**5s**	**10s**	**18s**	**32s**	**54s**	**1m24s**
bdd	**0s**	**1s**	**3s**	**5s**	**10s**	**18s**	33s	55s	1m30s
nodes	13	14	15	16	17	18	19	20	avg
monolithic	**2m11s**	3m33s	5m9s	7m18s	10m11s	14m0s	19m13s	**25m17s**	5m20s
cdnf	2m13s	**3m20s**	**4m54s**	**7m8s**	**10m10s**	13m52s	**19m5s**	25m57s	**5m18s**
bdd	2m16s	3m27s	5m3s	7m21s	**10m10s**	**13m50s**	19m11s	26m4s	5m21s

Fig. 6. Experimental Results for Synchronous Bus Arbiters

7 Conclusion

Using two exact learning algorithms for Boolean functions, the performance of assume-guarantee reasoning is compared against the monolithic interpolation-based Model Checking algorithm in this paper. Our experiments show that automated assume-guarantee reasoning through implicit learning can outperform monolithic Model Checking in some parametrized hardware test cases. For the OBDD learning algorithm, we demonstrate a simple technique to invert queries from the learning algorithms. The technique improves the performance of the OBDD learning algorithm in assume-guarantee reasoning.

In three of the five test cases, assume-guarantee reasoning significantly improves the average verification time. The compositional technique slightly outperforms monolithic Model Checking in the remaining two test cases. Between the two learning algorithms, the difference however is marginal except the property `m0s1m1` in the MSI protocol. The OBDD learning algorithm clearly dominates the CDNF algorithm in this case. On the other hand, the CDNF algorithm performs slightly better than the OBDD learning algorithm in all other cases.

The simple modification of the OBDD learning algorithm shows that learning algorithms may not be trivially applied in the learning-based approach to assume-guarantee reasoning. Further optimizations are needed in this application domain. Particularly, the performances of the CDNF algorithm and the OBDD learning algorithm are sensitive to variable orderings. More research on this regard may further improve the scalability of assume-guarantee reasoning.

References

1. Angluin, D.: Learning regular sets from queries and counterexamples. Information and Computation 75(2), 87–106 (1987)
2. Bryant, R.: Graph-based algorithms for Boolean-function manipulation. IEEE Transaction on Computers C-35(8) (1986)
3. Bshouty, N.H.: Exact learning boolean function via the monotone theory. Information and Computation 123(1), 146–153 (1995)
4. Campos, S.V.A., Clarke, E.M., Marrero, W.R., Minea, M.: Verifying the performance of the PCI local bus using symbolic techniques. In: ICCD, pp. 72–78 (1995)

5. Cantin, J.F., Lipasti, M.H., Smith, J.E.: Dynamic verification of cache coherence protocols. In: Workshop on Memory Performance Issues (2001)
6. Chaki, S., Strichman, O.: Optimized L^*-based assume-guarantee reasoning. In: Grumberg, O., Huth, M. (eds.) TACAS 2007. LNCS, vol. 4424, pp. 276–291. Springer, Heidelberg (2007)
7. Chen, Y.F., Clarke, E.M., Farzan, A., Tsai, M.H., Tsay, Y.K., Wang, B.Y.: Automated assume-guarantee reasoning through implicit learning. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174. Springer, Heidelberg (2010)
8. Chen, Y.F., Farzan, A., Clarke, E.M., Tsay, Y.K., Wang, B.Y.: Learning minimal separating DFA's for compositional verification. In: Kowalewski, S., Philippou, A. (eds.) TACAS. LNCS, vol. 5505, pp. 31–45. Springer, Heidelberg (2009)
9. Cimatti, A., Clarke, E., Giunchiglia, F., Roveri, M.: NuSMV: a new Symbolic Model Verifier. In: Halbwachs, N., Peled, D.A. (eds.) CAV 1999. LNCS, vol. 1633, pp. 495–499. Springer, Heidelberg (1999)
10. Clarke, E.M., Kröning, D.: SMV example: Bus protocol, PowerPoint file (2002)
11. Cobleigh, J.M., Giannakopoulou, D.: Păsăreanu, C.S.: Learning assumptions for compositional verification. In: Garavel, H., Hatcliff, J. (eds.) TACAS 2003. LNCS, vol. 2619, pp. 331–346. Springer, Heidelberg (2003)
12. Gavaldà, R., Guijarro, D.: Learning ordered binary decision diagrams. In: Zeugmann, T., Shinohara, T., Jantke, K.P. (eds.) ALT 1995. LNCS, vol. 997, pp. 228–238. Springer, Heidelberg (1995)
13. Gupta, A., McMillan, K.L., Fu, Z.: Automated assumption generation for compositional verification. Formal Methods in System Design 32(3), 285–301 (2008)
14. Handy, J.: The Cache Memory Book. Academic Press, London (1998)
15. Jung, Y., Kong, S., Wang, B.Y., Yi, K.: Deriving invariants in propositional logic by algorithmic learning, decision procedure, and predicate abstraction. In: Barthe, G., Hermenegildo, M. (eds.) VMCAI 2010. LNCS, vol. 5944, pp. 180–196. Springer, Heidelberg (2010)
16. Kimura, S., Clarke, E.M.: A parallel algorithm for constructing binary decision diagrams. In: ICCD, pp. 220–223 (1990)
17. McMillan, K.L.: The SMV system, symbolic model checking - an approach. Technical Report CMU-CS-92-131, Carnegie Mellon University (1992)
18. McMillan, K.L.: Interpolation and SAT-based model checking. In: Hunt Jr., W.A., Somenzi, F. (eds.) CAV 2003. LNCS, vol. 2725, pp. 1–13. Springer, Heidelberg (2003)
19. Nakamura, A.: An efficient query learning algorithm for ordered binary decision diagrams. Information and Computation 201(2), 178–198 (2005)
20. Nam, W., Madhusudan, P., Alur, R.: Automatic symbolic compositional verification by learning assumptions. Formal Methods in System Design 32(3), 207–234 (2008)
21. Sinha, N., Clarke, E.M.: SAT-based compositional verification using lazy learning. In: Damm, W., Hermanns, H. (eds.) CAV 2007. LNCS, vol. 4590, pp. 39–54. Springer, Heidelberg (2007)

Inferring Compact Models of Communication Protocol Entities

Therese Bohlin[1], Bengt Jonsson[1], and Siavash Soleimanifard[2]

[1] Dept. of Information Technology, Uppsala University, Sweden
{thereseb,bengt}@it.uu.se
[2] Royal Institute of Technology, Stockholm, Sweden
siavashs@csc.kth.se

Abstract. Our overall goal is to support model-based approaches to verification and validation of communication protocols by techniques that automatically generate models of communication protocol entities from observations of their external behavior, using techniques based on regular inference (aka automata learning). In this paper, we address the problem that existing regular inference techniques produce "flat" state machines, whereas practically useful protocol models structure the internal state in terms of control locations and state variables, and describes dynamic behavior in a suitable (abstract) programming notation. We present a technique for introducing structure of an unstructured finite-state machine by introducing state variables and program-like descriptions of dynamic behavior, given a certain amount of user guidance. Our technique groups states with "similar control behavior" into control locations, and obtain program-like descriptions by means of decision tree generation. We have applied parts of our approach to an executable state machine specification of the Mobile Arts Advanced Mobile Location Center (A-MLC) protocol and evaluated the results by comparing them to the original specification.

1 Introduction

Model-based techniques for verification, testing, and validation of commmunication protocols, including model checking and model-based testing [8], have witnessed drastic advances in the last decades. They require access to a formal model that specifies the behavior of protocol entities, which ideally should be developed during specification and design. However, the construction of models typically requires significant manual effort, implying that in many cases no such model is available, or becomes outdated as the system evolves over time. It is therefore important to develop automated techniques that support the task of producing models, e.g., models that reflect the behavior of an existing protocol implementation. Such techniques would be highly useful for producing models of standardized protocols, for introducing model based testing techniques to replace manual testing of an existing product, for regression testing, etc. A potential approach is to use program analysis to construct models from source code (e.g., [4,16]). However,

T. Margaria and B. Steffen (Eds.): ISoLA 2010, Part I, LNCS 6415, pp. 658–672, 2010.

many system components, such as library modules, or third-party components, often do not allow analysis of source code. We will therefore focus on techniques for constructing models from observations of their external behavior.

The construction of models from observations of component behavior can be performed using regular inference (aka automata learning) techniques [2,12,18,27]. This class of techniques has recently started to get attention in the testing and verification community, e.g., for regression testing of telecommunication systems [15,17], for integration testing [19,19,14], and for combining conformance testing and model checking [23,13]. In regular inference, a finite-state machine (or a regular language) is constructed from the answers to a set of *membership queries*, each of which observes the component's output in response to a certain input string. Given "enough" membership queries, the constructed automaton will be a correct model of the observed component.

Our overall goal is to construct models of entities in communication protocols, which can be readily understood and maintained by protocol designers and test engineers. Manually constructed models of protocol behavior facilitate understanding by describing messages as consisting of a message type with a number of parameters, by representing the internal states of the entity in terms of control locations and state variables, and by describing the reaction to incoming messages by a change of location and variable transformation in some suitable language. This style of modeling is supported by several formalisms, such as UML state diagrams [11].

A serious obstacle to constructing structured models from observations is that existing regular inference techniques produce "flat" state machines, in which neither states nor transitions have any structure. In this paper, we therefore present techniques for restructuring the representation of an unstructured finite-state machine, in order to make it readily understandable by humans. Since there are many ways to restructure state-machine descriptions, and since most likely there is no unique optimal restructuring, our techniques use some heuristically motivated general principles for forming state variables and control locations, which if needed can be changed by a user. Based on such principles, our transformation first equips the "flat" state machine with state variables. Thereafter it groups states with similar control behavior into control locations. Finally, the "flat" description of the reaction to received messages is transformed into a compact description in the chosen coding language; we have chosen the intuitive formalism of decision trees, which can be generated by well-developed tools.

We evaluate our techniques by applying them to the Mobile Arts Advanced Mobile Location Center (A-MLC) protocol, which is a commercially available middleware protocol that allows mobile network operators to provide presence information from the GSM/UMTS network. We have access to an executable specification of A-MLC, which is structured for human readability by developers and testers of the protocol. This makes it a suitable object for evaluation, since we can both observe its reaction to a large number of input sequences, as well as compare the results of our restructuring to the structure of the executable specification. We present the results of our comparison.

Related Work. Regular inference techniques have been used for verification and test generation, e.g., to create models of environment constraints with respect to which a component should be verified [10], for regression testing to create a specification and a test suite [15,17], to perform model checking without access to source code or formal models [13,23], for program analysis [1], and for formal specification and verification [10]. Groz, Li, and Shahbaz extend regular inference to Mealy machines with a finite subset of input and output symbols from the possible infinite set of symbols [19,28,14]. Mariani and Pezzé use inference in integration testing of commercial off the shelf components [20]. They infer two separate models: one for the finite-state control, and the others being a relation on the parameters in each interaction. They use different inference techniques for each type of model. In previous work [5], we presented an optimization of regular inference to cope with models where the domains of the parameters are booleans. We have also presented an approach using regular inference, in which systems have input parameters from a potentially infinite domain and parameters may be stored in state variables for later use [6].

Organization of Paper. In next section, we review Mealy machines and Symbolic Mealy Machines. In Section 3 we describe the employed inference algorithm for Mealy machines by Niese [22], we present our transformation from "flat" to symbolic Mealy machines. Our implementation is described in Section 4, and in Section 5 we describe its application so the A-MLC protocol, which is evaluated in Section 6. Section 7 contains conclusions and proposed future work.

2 Mealy Machines

Basic Definitions. We will use *Mealy machines* to model communication protocol entities. A *Mealy machine* is a tuple $\mathcal{M} = \langle \Sigma_I, \Sigma_O, Q, q_0, \delta, \lambda \rangle$ where Σ_I is a nonempty set of *input symbols*, Σ_O is a nonempty set of *output symbols*, Q is a nonempty set of *states*, $q_0 \in Q$ is the *initial state*, $\delta : Q \times \Sigma_I \to Q$ is the *transition function*, and $\lambda : Q \times \Sigma_I \to \Sigma_O$ is the *output function*. The sets of states and symbols can be finite or infinite: if they are all finite we say that the Mealy machine is *finite*. Elements of Σ_I^* are called *input strings*, and elements of Σ_O^* are called *output strings*. We extend the transition and output functions to input strings in the standard way, by defining:

$$\begin{array}{ll} \delta(q, \varepsilon) = q & \lambda(q, \varepsilon) = \varepsilon \\ \delta(q, ua) = \delta(\delta(q, u), a) & \lambda(q, ua) = \lambda(q, u)\lambda(\delta(q, u), a) \end{array}$$

where $u \in \Sigma_I^*$. We define $\lambda_{\mathcal{M}}(u) = \lambda(q_0, u)$ for $u \in \Sigma_I^*$. Two Mealy machines $\mathcal{M}$ and $\mathcal{M}'$ with the same set of input symbols are *equivalent* if $\lambda_{\mathcal{M}}(u) = \lambda_{\mathcal{M}'}(u)$ for all input strings u.

Intuitively, a Mealy machine behaves as follows. At any point in time, the machine is in some state $q \in Q$. When supplied with an input symbol $a \in \Sigma_I$, it responds by producing an output symbol $\lambda(q, a)$ and transforms itself to a

new state $\delta(q,a)$. We use the notation $q \xrightarrow{a/b} q'$ to denote that $\delta(q,a) = q'$ and $\lambda(q,a) = b$; in this case $q \xrightarrow{a/b} q'$ is called a *transition* of $\mathcal{M}$.

The Mealy machines that we consider are *deterministic*, meaning that for each state q and input symbol a exactly one next state $\delta(q,a)$ and output string $\lambda(q,a)$ is possible.

Symbolic Representation. In order to conveniently model entities of communication protocols, we should be able to describe messages as consisting of a message type with a number of parameters, describe states as consisting of a control location and values of a set of state variables, and describe the reaction to incoming messages in a suitable programming language-like syntax. We therefore introduce a symbolic representation of Mealy machines, similar to Extended Finite State Machines [24].

So, assume a set of *action types*. Each action type α has a certain *arity*, which is a tuple of *domains* (a domain is a set of allowed data values) $\mathcal{D}_{\alpha,1}, \ldots, \mathcal{D}_{\alpha,n}$ (where n depends on α). For a set I of action types, let Σ_I be the set of terms of form $\alpha(d_1, \ldots, d_n)$, where $d_i \in \mathcal{D}_{\alpha,i}$ is a data value in the appropriate domain for each i with $1 \leq i \leq n$. We write $\overline{d}$ for $d_1, \ldots, d_n$. Also assume a set of *state variables*. Each state variable has a domain of possible values, and a unique initial value. We use v to range over state variables, and $\mathtt{v}$ to range over values. We write $\overline{v}$ for $v_1, \ldots, v_k$ and $\overline{\mathtt{v}}$ for $\mathtt{v}_1, \ldots, \mathtt{v}_k$

To provide a structured representation of the transition and output functions, we use a simple formalism with constructs for selection, output, and assignment. We will use a finite set of *formal parameters*, ranged over by $p_1, p_2, \ldots$, which will serve as local variables to which values of parameters in input symbols are bound. We write $\overline{p}$ for $p_1, \ldots, p_n$. Let an *expression*, ranged over by e, be either a formal parameter, a state variable or a data value. Define the set of *action expression*, ranged over by ae, by the grammar

$$\begin{aligned} oe &::= \textbf{output}\ \beta(d_1, \ldots, d_m);\ \textbf{nextloc}\ l \\ &\mid \textbf{case}\ e\ \textbf{of}\ d_1\ :\ oe_1\ \ \cdots\ \ d_k\ :\ oe_k \\ ae &::= oe\ ;\ v_1, \ldots, v_k := e_1, \ldots, e_k \end{aligned}$$

Here, $v_1, \ldots, v_k := e_1, \ldots, e_k$ simultaneously assigns the values of the expressions e_i to the variables v_i. In a **case** expression, the values $d_1, \ldots, d_k$ must be different, and cover the possible results of evaluating the expression e. Sometimes we use **if** e **then** ae_1 **else** ae_2 instead of **case** e **of** true : ae_1 false : ae_2.

Intuitively, an action expression first traverses a decision tree. Depending on the values of state variables and input parameter, eventually an output symbol is generated, and the next control location is determined. Before actually changing control location, the state variables are updated in a multiple assignment statement.

Definition 1. *A* Symbolic Mealy machine (SMM) *is a tuple* $\mathcal{SM} = \langle I, O, L, l^0, \overline{v}, \varphi \rangle$*, where* I *and* O *are disjoint finite sets of action types* (input action types *and* output action types)*, where* L *is a finite set of* locations*, where* $l^0 \in L$ *is the*

initial location, *where* $\overline{v}$ *is a finite tuple* $v_1, \ldots, v_k$ *of state variables, and where* φ *maps each location* $l \in L$ *and input action type* $\alpha \in I$ *to an action expression.*

We write **in location** l **when** $\alpha(p_1, \ldots, p_m)$ ae **end** to denote that $\varphi(l, \alpha)$ is the action expression ae.

The meaning of a SMM $\mathcal{SM} = \langle I, O, L, l^0, \overline{v}, \longrightarrow \rangle$ is defined by its denotation, which is the Mealy machine $\mathcal{M}_{\mathcal{SM}} = \langle \Sigma_I, \Sigma_O, Q, q_0, \delta, \lambda \rangle$, where Σ_I is obtained from I as described earlier, and similarly for Σ_O, where Q is the set of pairs $\langle l, \overline{\mathtt{v}} \rangle$ consisting of a location $l \in L$ and tuple $\overline{\mathtt{v}}$ of values of the state variables $\overline{v}$, where q_0 is the pair $\langle l^0, \mathtt{v}_1^0, \ldots, \mathtt{v}_k^0 \rangle$ in which $\mathtt{v}_1^0, \ldots, \mathtt{v}_k^0$ are the initial values of $v_1, \ldots, v_k$.

The reaction to input symbols is described by the mapping φ, as follows. For each location $l \in L$ and input symbol $\alpha(\overline{d})$, the action expression $\varphi(l, \alpha)$ will follow exactly one branch of the nested **case** expression leading to an expression of form **output** $\beta(\overline{d}')$; **nextloc** l'; thereafter follows a multiple assignment of form $v_1, \ldots, v_k := e_1, \ldots, e_k$. This implies that for the transition and output functions we have $\delta(\langle l, \mathtt{v}_1, \ldots, \mathtt{v}_k \rangle, \alpha(\overline{d})) = \langle l', \mathtt{v}'_1, \ldots, \mathtt{v}'_k \rangle$, and $\lambda(\langle l, \mathtt{v}_1, \ldots, \mathtt{v}_k \rangle, \alpha(\overline{d})) = \beta(\overline{d}')$, for all tuples $\mathtt{v}'_1, \ldots, \mathtt{v}'_k$ of values of $v_1, \ldots, v_k$, where $\mathtt{v}'_1, \ldots, \mathtt{v}'_k$ is the result of performing the multiple assignment statement.

In Figure 1, we show a possible action expression, from an idealized version of the receiver in the alternating bit protocol, in which we have action types `Data` and `Ack`, each of which has a bit (either 0 or 1) as a parameter.

```
in location rec when Data(sn)
    case (sn) of  0 :  case v of  0 : output Ack(0); nextloc rec;
                                  1 : output Ack(1); nextloc rec;
                  1 :  case v of  1 : output Ack(1); nextloc rec;
                                  0 : output Ack(0); nextloc rec;
      v := sn
end
```

Fig. 1. Example syntax defining part of reciever in alternating bit protocol

3 Inference of Symbolic Mealy Machines

In this section, we present our approach for inferring an SMM model for the behavior of an entity in a communication protocol, by observing its responses to selected input strings. We will hereafter refer to the given protocol entity as the System Under Test (SUT). We assume that the behavior of a SUT can be modeled as an SMM, and that its input and output action types as well as their arities, are known.

The problem of inferring a model of the SUT naturally decomposed into two subproblems. First we infer a "flat" Mealy machine $\mathcal{M}$ which models the behavior of SUT. Thereafter, we generate an SMM $\mathcal{SM}$ such that $\mathcal{M}_{\mathcal{SM}}$ is equivalent to $\mathcal{M}$. For the first subproblem we use an adaptation of the L^* algorithm [2] to

Mealy machines, due to Niese [22]. For the second subproblem, we have developed a technique for transforming a Mealy machine into an SMM by introducing state variables, control locations, and action expressions. Each subproblem is described in more detail in the following subsections.

3.1 Inference of Mealy Machines

To infer a Mealy machine that models the behavior of SUT, we use an adaptation of the L^* algorithm due to Niese [22]. It is assumed that the L^* algorithm initially knows the static interface of $\mathcal{SM}$, i.e., the sets I and O of input and output actions together with their arities. It may then ask a sequence of *membership queries*; each one supplying a chosen input string $u \in (\Sigma_I)^*$ and observing the response $\lambda_{\mathcal{SM}}(u)$. After a "sufficient" number of membership queries the *Learner* can build a "stable" hypothesis $\mathcal{H}$ from the obtained information. The hypothesis $\mathcal{H}$ should of course agree with $\mathcal{SM}$ on the performed membership queries (i.e., $\lambda_{\mathcal{SM}}(u) = \lambda_{\mathcal{H}}(u)$ whenever u was supplied in a membership query), but must make suitable generalizations for other input strings. In order to increase confidence in the hypothesis $\mathcal{H}$, one can subject $\mathcal{SM}$ to thorough conformance testing or longer-term monitoring in order to search for input strings on which $\mathcal{SM}$ disagrees with $\mathcal{H}$. In the setting of L^*, this is idealized as an *equivalence query*, which asks whether $\mathcal{H}$ is equivalent to $\mathcal{SM}$, and which is replied with either *yes*, or with *no* and a *counterexample*, which is an input string $u \in \Sigma_I^*$ such that $\lambda_{\mathcal{SM}}(u) \neq \lambda_{\mathcal{H}}(u)$. In a black-box setting, where source code is not available, there is in general no perfect implementation of equivalence queries. In the case that there is a known upper bound on the number of states of $\mathcal{M}$, (typically large) conformance test suites (as described in, e.g., [9,29]) can conclusively settle equivalence queries. In practice, equivalence queries are often approximated by large random test suites and/or by monitoring the SUT under a long period of time. The algorithm is guaranteed to terminate after at most n such equivalence queries, where n is the number of states of $\mathcal{M}$, having posed in total $O(|\Sigma_I|n^2 + n \log m)$ membership queries, where m is the length of the longest counterexample returned in some equivalence query [27].

3.2 Generating Symbolic Representation of Mealy Machines

In this subsection, we describe our transformation from a Mealy machine $\mathcal{M}$ into an equivalent SMM $\mathcal{SM}$ (i.e., such that $\mathcal{M}_{\mathcal{SM}}$ is equivalent to $\mathcal{M}$), which can more easily be understood by human designers or testers. The transformation (1) represents the states of $\mathcal{M}$ in terms of control locations and state variables, and (2) represents the transition and output function of $\mathcal{M}$ in terms of action expressions. We first describe how we introduce a symbolic representation of states, and thereafter how we generate action expressions.

Transforming the Representation of States. In the symbolic representation, states are formed as a combination of control locations that capture "high level control" aspects of behavior, and of state variables that record information in received parameters of input symbols that may influence future behavior. For

a given "flat" Mealy machine, there are several ways to accomplish this, among which there is most likely no "best" one. Our transformation makes default choices in the following way.

- Sequences of transitions that contain the same sequence of input and output action types should lead to the same control location. For example, by applying this criterion, the Mealy machine described in Figure 1 would be transformed into an SMM in which only one control location could be reached from the initial location, since there is only one combination of input and output action types (namely `Data/Ack`).
- For each input action type α with arity $\mathcal{D}_{\alpha,1}, \ldots, \mathcal{D}_{\alpha,n}$ there are state variables $v_{\alpha,1}, \ldots, v_{\alpha,n}$ which record the values of the parameters in the most recently received input symbol of form $\alpha(d_1, \ldots, d_n)$. The transformation chooses default initial values for these variables. With this principle, the Alternating bit protocol in Figure 1 would have a state variable $v_{\texttt{Data.sn}}$, which is assigned the parameter value `sn` in action expressions triggered by the action type `Data`.

In our implementation, these default choices can be replaced by other criteria for forming state variables and control locations. To keep the presentation in this section simple, they are briefly described in Section 4.

Using the above principles, our transformation generates a symbolic representation of states as follows. Let an *extended state* be defined as a pair $\langle q, \overline{\mathrm{v}} \rangle$, where $q \in Q$ is a state of $\mathcal{M}$ and $\overline{\mathrm{v}}$ is a tuple of values of the state variables $\overline{v}$. Thus, for each state q of $\mathcal{M}$, there are many extended states of form $\langle q, \overline{\mathrm{v}} \rangle$, corresponding to the many different combinations of values $\overline{\mathrm{v}}$ that may be received along different execution paths. Let an *extended transition* be a transition of form $\langle q, \overline{\mathrm{v}} \rangle \xrightarrow{\alpha(\overline{d})/\beta(\overline{d}')} \langle q', \overline{\mathrm{v}}' \rangle$ between extended states, which exists whenever $\mathcal{M}$ has a transition $q \xrightarrow{\alpha(\overline{d})/\beta(\overline{d}')} q'$ and $\overline{\mathrm{v}}'$ is obtained from $\overline{\mathrm{v}}$ and $\alpha(\overline{d})$ by appropriately updating state variables.

We must now form control locations as sets of extended states with "same control behavior", using a technique similar to the subset construction for nondeterministic finite automata. Such an algorithm is described in Algorithm 1. The algorithm maintains two sets of locations; *Locs* accumulates the set of formed locations, whereas *TempLocs* is a set of locations whose successor locations remain to be constructed, and a set *Edges* of generated edges. Algorithm 1 starts by forming the initial location $l^0 \in L$, containing the extended state formed from the initial state q_0 and initial values $\overline{\mathrm{v}}_0$ of variables. The algorithm then iteratively picks some location l from *TempLocs*; for each pair α, β of input and output action types it constructs a new location containing the targets of all transitions $\langle q, \overline{\mathrm{v}} \rangle \xrightarrow{\alpha(\overline{d})/\beta(\overline{d}')} \langle q', \overline{\mathrm{v}}' \rangle$ with the same source location, input, and output action types, and adds it to *TempLocs*, and also adds $l \xrightarrow{\alpha/\beta} l'$ to *Edges*. The process of forming locations continues iteratively until all locations in *TempLocs* have been used for forming successor locations. The process is guaranteed to terminate since the set of extended states is finite.

Algorithm 1. MAKELOCATIONS

1: $Locs := \emptyset$;
2: $Edges := \emptyset$;
3: $TempLocs := \{\langle q_0, \overline{\mathrm{v}}_0 \rangle\}$;
4: **while** $TempLocs \neq \emptyset$ **do**
5: **choose** $l \in TempLocs$;
6: **for all** pairs α, β **do**
7: $l' := \{\langle q', \overline{\mathrm{v}}' \rangle \ : \ \exists \langle q, \overline{\mathrm{v}} \rangle \in l \ , \exists \overline{d}, \overline{d}' . \ \langle q, \overline{\mathrm{v}} \rangle \xrightarrow{\alpha(\overline{d})/\beta(\overline{d}')} \langle q', \overline{\mathrm{v}}' \rangle\}$;
8: **if** ($l' \neq \emptyset$ **and** $l' \notin (Locs \cup TempLocs)$) **then**
9: $TempLocs := TempLocs \cup \ l'$;
10: $Edges := Edges \cup \ l \xrightarrow{\alpha/\beta} l'$;
11: **end if**
12: **end for**
13: $TempLocs := TempLocs \setminus \ l$;
14: $Locs := Locs \cup \ l$;
15: **end while**

During Algorithm 1, we additionally merge locations which are "similar", in the sense that they share an extended state, since presumably their future behavior is rather similar. Such new formed locations are added to *TempLocs* to properly generate their successors. However, we must not merge locations if as a result they will contain two extended states $\langle q, \overline{\mathrm{v}} \rangle$ and $\langle q', \overline{\mathrm{v}} \rangle$ with the same variable values but different control state, since action expressions (which can only test values of variables) will not be able to distinguish the difference in future behavior between q and q'.

Generating Action Expressions. It remains to generate an action expression for each location l and input action type α, which distinguishes between the different behaviors of different extended states in the location. Our transformation generates action expressions as decision tree structures of **case** expressions, each of which tests some input parameters in $p_1, \ldots, p_n$ or state variable in $\overline{v}$, reaching appropriate leaves of form **output** $\beta(\overline{d}')$; **nextloc** l' . By thereafter adding the appropriate assignment statement, a complete action expression has been generated.

The decision tree structure of the **case** expressions in the action expression of location l and input action type α should be constructed so that whenever it is presented with values $\overline{d}$ of input parameters $\overline{p}$ and values $\overline{\mathrm{v}}$ of state variables $\overline{v}$, such that

$$\langle q, \overline{\mathrm{v}} \rangle \xrightarrow{\alpha(\overline{d})/\beta(\overline{d}')} \langle q', \overline{\mathrm{v}}' \rangle$$

is an extended transition from some $\langle q, \overline{\mathrm{v}} \rangle \in l$ to some $\langle q', \overline{\mathrm{v}}' \rangle \in l'$ with $l \xrightarrow{\alpha/\beta} l' \in Edges$, then the decision tree should reach the output symbol $\beta(\overline{d}')$ and location l'. There are well-developed algorithms to generate decision trees from a set of such constraints, among the most well-known being ID3 [25,21]. The ID3

algorithm generates a minimal decision tree from a given set of examples (in our case generated from extended transitions as above). The generated decision tree structures are typically much more compact than the set of "flat" Mealy machine transitions that they cover, in particular if the input alphabet is large.

4 Implementation

Based on the techniques described in Section 3.2, we have developed an implementation, which gets a description of a "flat" Mealy machine, possibly together with user-supplied criteria for forming state variables and control locations to override the default ones, and generates a Symbolic Mealy machine. Several non-default criteria for making control locations and/or decision trees have been considered and implemented. In addition to the criterion used in Algorithm 1, our implementation also accepts the criterion that sucessor locations of extended transitions $\langle q, \overline{v}\rangle \xrightarrow{\alpha(\overline{d})/\beta(\overline{d}')} \langle q', \overline{v}'\rangle$ with the same output action type β should be in the same location, as well as the criterion that extended transitions with the same output symbol $\beta(\overline{d}')$ should be in the same location. For these criteria, Algorithm 1 and the generation of action expressions are changed accordingly. Users can try alternative criteria to see which resulting structure better suits their purpose.

In our tool we use an implementation of ID3 provided by Weka (Waikato Environment for Knowledge Analysis) a data mining tool developed at the University of Waikato, New Zealand, and distributed under the Gnu Public Licence. It includes a wide variety of state-of-the-art algorithms of data mining and machine learning which are implemented in Java [30].

5 Experiments

In this section, we describe the application of our implemented technique to generating a model of the Mobile Arts Advanced Mobile Location Center (A-MLC) protocol. We have chosen A-MLC because we have access to an executable specification, which has been created to be understood by developers and testers. This makes A-MLC suitable for our experimentation since we can both execute large numbers of membership queries and can compare our resulting model with the provided one.

The A-MLC protocol is a middle-ware product that allows Mobile Network Operators to provide presence information from the GSM/UMTS network, including details about the location, present status, and capabilities of mobile devices. It is commercially available and has been deployed at several telecom operators within Europe.

The implementation of A-MLC was made mainly in Erlang [3] It consists of approximately 130,000 lines of Erlang code and 5,500 lines of C code. The originators of the A-MLC protocol have written a functional specification of the protocol in order to generate high-quality test suites [7]. The specification

essentially has the form of a Symbolic Mealy machine, and captures all traffic sequences through A-MLC. Low level protocols, as well as operation and maintenance interfaces, are not part of the specification. The specification models the behavior resulting from an individual client request, since the interaction between concurrent requests is minimal.

We have used an executable version of this specification, implemented using the Erlang behavior module *gen_fsm* (Generic Finite State Machine Behavior), as the SUT. The specification consists of 13 control states, 23 state variables, and 10 input action types with different arities.

For the inference experiment, we defined small domains for the values of parameters in input symbols, in order to be able to carry out enough membership queries to complete the inference process. For most parameters, these domains were already small in the original specification (typically $2-4$ values), and for others, we could choose a representive sample that would allow coverage of the entire specification. In one case, however, this reduction made a part of the model unreachable (as described in Section 6): for input parameter `status` of `atir` action type which can assume values `not_reachable`, `reachable` and `undefined`, we only used the value `not_reachable`. In all, this resulted in an input alphabet of 1560 input symbols.

To construct a Mealy machine model of the executable specification of the A-MLC protocol by regular inference, we used the LearnLib tool [26], developed at the University of Dortmund, which has an efficient implementation of the L^* algorithm. This tool provides several different realizations of equivalence queries, including conformance tests suite generated by the Vasilevsky-Chow algorithm [9,29]), and random test suites of user-controlled size. We finally applied our implementation of the transformation in 3.2 to transform the flat Mealy machine generated by LearnLib into a symbolic one.

6 Results

Applying LearnLib to the executable specification resulted in a Mealy machine with 44 states. It took about 43 hours to complete the inference, during which LearnLib asked about 175 million membership queries. As equivalence oracle, LearnLib used a test suite of 1000 randomly generated tests of length 10.

The generated "flat" Mealy machine exhibits 59 different sequences of output symbols on its transitions, formed from 11 distinct sequences of output action types. A part of the Mealy machine is shown in Figure 2. In the figure, states $q0$ till $q8$ can be seen with some of their transitions, which we have labeled by pairs of input-output action types. E.g., the dashed transition from state $q7$ to state $q1$, has `srir` as input action type and `write_cache` and `slia` as output action types. Later in this section, in Table 3a, we correlate the states of this figure with control locations of the executable specification.

Most of the transitions of the Mealy machine output the error symbol (representing that the corresponding input symbol is "illegal"). Before generating a symbolic representation using our tool, we removed these, since we are interested

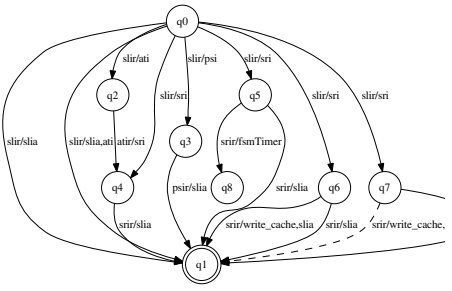
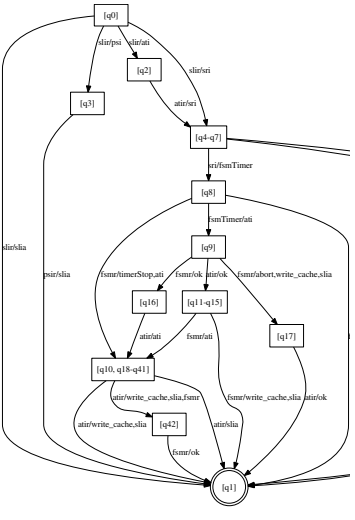


Fig. 2. The inferred Mealy machine, states $q0 - q8$

Location	Control State
$[q0]$	IDLE
$[q1]$	DONE
$[q2]$	LAST_POS
$[q3]$	LAST_POS
$[q4\text{-}q7]$	ACCESS_NETPARAM
$[q8]$	FORCE_UPDATE
$[q9]$	TIMER_TRIGGERED
$[q11\text{-}q15]$	NOT_YET_UPDATED, WAIT_FSM_RESP
$[q16]$	MAYBE_UPDATED
$[q17]$	WAIT_POS_RESP
$[q10, q18\text{-}q41]$	UPDATED
$[q42]$	TERMINATE_MMS
	LAST_NETPARAM

(a) Correspondence between locations in our symbolic representation and in the executable specification



(b) The compacted structure

Fig. 3.

in being equivalent with respect to the legal input strings. The structure of control locations and edges generated by our transformation is shown in Figure 3b, We used the same criterion for forming control locations as used in Algorithm 1. In Figure 3b, boxes represent locations. Each location is labeled with the set of states of that "flat" Mealy machine that occured in forming this location.

6.1 Evaluation

To evaluate our transformation we compare

– *coverage*: the number of the control locations and edges in the executable specification that are captured in our symbolic representation,

- *similarity*: of the locations in our symbolic representation and of those in the executable specification,
- *readability*: of the action expressions of our symbolic representation, as compared with those in the executable specification.

Coverage. Out of the 13 control locations of the executable specification, 12 have been reached in our symbolic representation. The control location LAST_NETPARAM could not be reached, since we had reduced the range of parameter `status`, as described in Section 5.

The executable specification has 60 edges. The described reduction of the parameter range of `status` causes 20 of these to become unreachable. Of the remaining 40, our model captured 26. The missing 14 edges are all missing for the reason that LearnLib incorrectly merged two particular states in the flat Mealy machine. Let us explain how. The state $q5$ in Figure 2 is reached by (among others) `slir` messages with both the values `psi` and `ati` of one particular parameter. The effect of these parameter values is not externally observable immediately in the behavior of the SUT, but shows up only two transitions later. However, the L^* algorithm sees that the message following the `slir` message with parameter value `psi` triggers the same output as the message following the `slir` message with parameter value `ati`. L^* then assumes that all the replies to all following messages does not depend on whether the parameter value `psi` or `ati` was supplied with the `slir` message. It then continues to explore continued behavior of the SUT only for longer input strings that start with the `ati` value. This problem can be avoided by having more powerful test suites in equivalence oracles. Our equivalence test used only 1000 randomly chosen input strings of length 10; we conjecture that longer input strings and a larger equivalence test would discover the differences between the two parameter values.

Similarity. Table 3a shows how the locations of our symbolic representation correspond to those of the executable specification. The locations NOT_YET_UPDATED and WAIT_FSM_RESP are not distinguished in our symbolic representation, since they are reached by the same sequence of input-output action types. Also, locations [q2] and [q3] correspond to location LAST_POS, which can be reached by two different pairs of input-output action types.

```
1  in location IDLE
2  when Slir(msis,loct,netp,epsi,frc,lra)
3    if(epsi)
4      if(frc)or((!frc)and((lra)and(loct = last)))
5        case (netp) of
6          false : output Psi(netp);nextloc LAST_POS;
7          true : output Sri(msis);nextloc ACCESS_NETPARAM;
8        endcase
9      else if((!frc)and(!lra)){output Slia(netp,msis);nextloc DONE;}
10     else{output ErrMsg;nextloc ErrLoc;}
11   ...
12   MSIS:=msis;LOCT:=loct;NETP:=netp;FRC:=frc;LRA:=lra;
13 end
```

Fig. 4. Small extract of executable specification

```
1  in location IDLE
2  when Slir(msis,loct,netp,epsi,frc,lra)
3    if (epsi)
4      case netp of
5        false :
6          if(!frc)
7            if (!lra) {output Slia(netp,msis);nextloc DONE;}
8            else if(lra) {output Psi(netp);nextloc LAST_POS;}
9          else if(frc) {output Psi(netp,msis);nextloc LAST_POS;}
10       true :
11         if(!frc) {
12           if (!lra) {output Slia(netp,msis);nextloc DONE;}
13           else if (lra) {output Sri(msis);nextloc ACCESS_NETPARAM;}
14         else if(frc) {output Sri(msis);nextloc ACCESS_NETPARAM;}
15     endcase
16   ...
17   MSIS:=msis;LOCT:=loct;NETP:=netp;FRC:=frc;LRA:=lra;
18 end
```

Fig. 5. Small extract of action expression related to specification part in Figure 4

Readability. Since we cannot compare the two models in their entirety, we have chosen to compare two typical action expressions, representing the same behavior. Figure 5 shows a part of our generated action expression from the initial location when a message of form `Slir` with formal parameters (`msis`, `loct`, `maxage`, `netp`, `epsi`) is received, and Figure 4 shows the corresponding part of the executable specification. In the figures, the values of input parameters are assigned to state variables, shown by upper-case letters, in line 17 of Figure 5 and line 12 of Figure 4. To simplify the comparison between the action expression and executable specification we have replaced the parameter values of output symbols by the names of parameters received with the input symbol. For this we carefully matched the values of the parameters in output symbols with the input action type's parameter names and found the corresponding parameter name for each parameter value.

We see that the action expression is more compact in the executable specification. One reason is that it uses complex boolean expressions (e.g., Figure 4 line 4), whereas our representation only uses a simple decision tree structure which tests one parameter or variable at a time. This makes the executable specification smaller than our representation, but sometimes more difficult to understand.

Another difference is that our representation does not explicitly return an error message on illegal input. This allows our action expressions to sometimes omit distinctions. In this example, the parameter `loct` is tested in Figure 4 line 4 but not in Figure 5.

7 Conclusions and Future Work

We have presented an technique for using regular inference to infer symbolic models of communication protocol entities, aiming to make them compact and readable. We first apply existing regular inference techniques to construct a "flat" Mealy machine model of the protocol, which is thereafter folded into a Symbolic Mealy machine. We have applied our approach to an executable specification of the A-MLC protocol developed by Mobile Arts. We used LearnLib to generate a

flat Mealy machine, which was then transformed into a symbolic representation by our implementation. We evaluated the result by comparing it to the original executable specification.

The two models had many similarities, but differed in some respects. Our model did not cover all the locations and transitions of the SUT, due to an incorrect merging of two states by the L^* algorithm, which caused a part of the behavior to be unexplored. We conjecture that this problem would go away if we would have used a larger test suite for checking the generated model; at the time of the experiment our time and space resources did not allow this. Our structure of locations was surprisingly similar to that of the manually generated executable specification. Our action expressions has a rather simple form, and thus they become longer than corresponding hand-generated ones. This suggests to look at more advanced ways to generate action expressions in a richer syntax, and to employ code transformations that reduce redundancies.

Acknowledgments. We are grateful to Harald Raffelt, Bernhard Steffen and Falk Howar for generous support when using the LearnLib tool, and for helpful discussions.

References

1. Ammons, G., Bodik, R., Larus, J.: Mining specifications. In: Proc. 29th ACM Symp. on Principles of Programming Languages, pp. 4–16 (2002)
2. Angluin, D.: Learning regular sets from queries and counterexamples. Information and Computation 75(2), 87–106 (1987)
3. Armstrong, J.: Programming ERLANG: software for a concurrent world. In: The Pragmatic Programmers (2007)
4. Ball, T., Rajamani, S.: The SLAM project: Debugging system software via static analysis. In: Proc. 29th ACM Symp. on Principles of Programming Languages, pp. 1–3 (2002)
5. Berg, T., Jonsson, B., Raffelt, H.: Regular inference for state machines with parameters. In: Baresi, L., Heckel, R. (eds.) FASE 2006. LNCS, vol. 3922, pp. 107–121. Springer, Heidelberg (2006)
6. Berg, T., Jonsson, B., Raffelt, H.: Regular inference for state machines using domains with equality tests. In: Fiadeiro, J.L., Inverardi, P. (eds.) FASE 2008. LNCS, vol. 4961, pp. 317–331. Springer, Heidelberg (2008)
7. Blom, J., Jonsson, B.: Automated test generation for industrial erlang applications. In: Proc. 2003 ACM SIGPLAN workshop on Erlang, Uppsala, Sweden, pp. 8–14 (August 2003)
8. Broy, M., Jonsson, B., Katoen, J.-P., Leucker, M., Pretschner, A. (eds.): Model-Based Testing of Reactive Systems. LNCS, vol. 3472. Springer, Heidelberg (2005)
9. Chow, T.S.: Testing software design modeled by finite-state machines. IEEE Trans. on Software Engineering 4(3), 178–187 (1978)
10. Cobleigh, J., Giannakopoulou, D., Pasareanu, C.: Learning assumptions for compositional verification. In: Garavel, H., Hatcliff, J. (eds.) TACAS 2003. LNCS, vol. 2619, pp. 331–346. Springer, Heidelberg (2003)
11. Fowler, M.: UML Distilled – A Bried Guide to the Standard Object Modeling Language, 3rd edn. Addison-Wesley, Reading (2008)

12. Gold, E.M.: Language identification in the limit. Information and Control 10(5), 447–474 (1967)
13. Groce, A., Peled, D., Yannakakis, M.: Adaptive model checking. In: Katoen, J.-P., Stevens, P. (eds.) TACAS 2002. LNCS, vol. 2280, pp. 357–370. Springer, Heidelberg (2002)
14. Groz, R., Li, K., Petrenko, A., Shahbaz, M.: Modular system verification by inference, testing and reachability analysis. In: Suzuki, K., Higashino, T., Ulrich, A., Hasegawa, T. (eds.) TestCom/FATES 2008. LNCS, vol. 5047, pp. 216–233. Springer, Heidelberg (2008)
15. Hagerer, A., Hungar, H., Niese, O., Steffen, B.: Model generation by moderated regular extrapolation. In: Kutsche, R.-D., Weber, H. (eds.) FASE 2002. LNCS, vol. 2306, pp. 80–95. Springer, Heidelberg (2002)
16. Henzinger, T., Jhala, R., Majumdar, R., Sutre, G.: Lazy abstraction. In: Proc. 29th ACM Symp. on Principles of Programming Languages, pp. 58–70 (2002)
17. Hungar, H., Niese, O., Steffen, B.: Domain-specific optimization in automata learning. In: Hunt Jr., W.A., Somenzi, F. (eds.) CAV 2003. LNCS, vol. 2725, pp. 315–327. Springer, Heidelberg (2003)
18. Kearns, M., Vazirani, U.: An Introduction to Computational Learning Theory. MIT Press, Cambridge (1994)
19. Li, K., Groz, R., Shahbaz, M.: Integration testing of distributed components based on learning parameterized I/O models. In. In: Najm, E., Pradat-Peyre, J.-F., Donzeau-Gouge, V.V. (eds.) FORTE 2006. LNCS, vol. 4229, pp. 436–450. Springer, Heidelberg (2006)
20. Mariani, L., Pezzè, M.: Dynamic detection of COTS components incompatibility. IEEE Software 24(5), 76–85 (2007)
21. Mitchell, T.M.: Machine Learning. McGraw-Hill, New York (1997)
22. Niese, O.: An integrated approach to testing complex systems. PhD thesis, Dortmund University (2003)
23. Peled, D., Vardi, M.Y., Yannakakis, M.: Black box checking. In: FORTE/PSTV 1999, Beijing, China, pp. 225–240. Kluwer, Dordrecht (1999)
24. Petrenko, A., Boroday, S., Groz, R.: Confirming configurations in EFSM testing. IEEE Trans. on Software Engineering 30(1), 29–42 (2004)
25. Quinlan, J.: Induction of decision trees. Machine Learning 1(1), 81–106 (1986)
26. Raffelt, H., Steffen, B., Berg, T.: Learnlib: a library for automata learning and experimentation. In: FMICS 2005, New York, NY, USA, pp. 62–71 (2005)
27. Rivest, R., Schapire, R.: Inference of finite automata using homing sequences. Information and Computation 103, 299–347 (1993)
28. Shahbaz, M., Li, K., Groz, R.: Learning and integration of parameterized components through testing. In: Petrenko, A., Veanes, M., Tretmans, J., Grieskamp, W. (eds.) TestCom/FATES 2007. LNCS, vol. 4581, pp. 319–334. Springer, Heidelberg (2007)
29. Vasilevski, M.P.: Failure diagnosis of automata. Cybernetic 9(4), 653–665 (1973)
30. Witten, I.H., Frank, E.: Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations. Morgan Kaufmann, San Francisco (1999)

Inference and Abstraction of the Biometric Passport*

Fides Aarts[1], Julien Schmaltz[1,2], and Frits Vaandrager[1]

[1] Institute for Computing and Information Sciences, Radboud University Nijmegen
{f.aarts,f.vaandrager}@cs.ru.nl
[2] School of Computer Science, Open University of The Netherlands
julien.schmaltz@ou.nl

Abstract. Model-based testing is a promising software testing technique for the automation of test generation and test execution. One obstacle to its adoption is the difficulty of developing models. Learning techniques provide tools to automatically derive automata-based models. Automation is obtained at the cost of time and unreadability of the models. We propose an abstraction technique to reduce the alphabet and large data sets. Our idea is to extract a priori knowledge about the teacher and use this knowledge to define equivalence classes. The latter are then used to define a new and reduced alphabet. The a priori knowledge can be obtained from informal documentation or requirements. We formally prove soundness of our approach. We demonstrate the practical feasibility of our technique by learning a model of the new biometric passport. Our automatically learned model is of comparable size and complexity of a previous model manually developed in the context of testing a passport implementation. Our model can be learned within one hour and slightly refines the previous model.

1 Introduction

Learning techniques, e.g., regular inference (also known as automata learning) [4], can be used to automatically create a model from an existing implementation. The regular inference algorithms provide sequences of inputs, called *membership queries*, to a system and observe the responses. In addition, *equivalence queries* check whether the procedure is completed. The practical application of learning techniques faces two issues: (1) the time to learn a model grows very fast with the size of the input alphabet and (2) automatically learned models are hard to read. Recently, a new abstraction technique has been proposed to reduce the size of the input alphabet and learn readable models [1,2].

Model-based testing (MBT) is a promising software testing approach providing full automation of test-cases generation and test-cases execution. Test-cases are automatically derived from a specification model of the System Under Test

* Supported by the European Community's 7th Framework Programme No. 214755 (QUASIMODO).

T. Margaria and B. Steffen (Eds.): ISoLA 2010, Part I, LNCS 6415, pp. 673–686, 2010.

(SUT). The MBT paradigm requires the existence of a formal model. Developing models is a complex, time-consuming, and error-prone task. Moreover, software systems evolve rapidly and models have to be updated or even new models have to be constructed. This cost of developing and maintaining models is a major obstacle to the wide adoption of MBT. Learning techniques could here play a role. In contrast to models, prototypes and partial implementations are always available during the development of software. One could learn a model from a reference implementation and use this model to test whether new implementations are still conforming to this reference model. Learning techniques could be used to derive the model at the first place.

In this paper, we apply the abstraction technique of Aarts et al. [1,2] to learn a model of the new generation of biometric passports [10,6]. The main idea of the abstraction technique is to extract a bit of *a priori* knowledge from documentation or interviews and use it to divide concrete parameter values into a small number of equivalence classes. This speeds up the learning process and reduces model size. In contrast to a previous application of this technique [1,2] we validate our automatically derived model against a previous hand-made specification of the passport [13]. This specification was used to validate the Dutch implementation of the biometric passport using the ioco-theory for MBT [16]. We implemented our abstraction as a mapping module and connected it to the LearnLib library for regular inference [15]. After translating our automatically derived Mealy machine to a Labelled Transition System (LTS), we used the tool JTorX [5] to show that this learned model is ioco-conforming to the hand-made specification. Our model can be learned within one hour and is of comparable complexity and readability as the hand-made one. It took several hours to develop the latter.

Our main contribution is to demonstrate and validate the applicability of our abstraction technique for learning automata to a practical and realistic case-study. The main result is that the model learned is comparable in size and correct w.r.t. to a previously hand-made specification. The time needed for a computer to learn the model from an existing implementation is much less than the time needed by a human to develop it.

The rest of the paper is organized as follows. In the next section, we give an overview of our approach. In Section 3, we review the Mealy machine model, regular inference, and our abstraction technique. Section 4 gives a short overview of the biometric passport; the experiments and according results are reported in Section 5. Finally, Section 6 contains conclusions and directions for future work.

2 Overview

Our approach works as follows. The goal is to learn a model of a *SUT* - the biometric passport. For the learning process we use three components: a *Learner* (LearnLib), a *Teacher* (*SUT*), and an intermediate layer called *Abstraction mapping* that reduces the alphabet of the *SUT*, see Learning box in Figure 1. The abstraction mapping is created using a priori knowledge extracted from informal

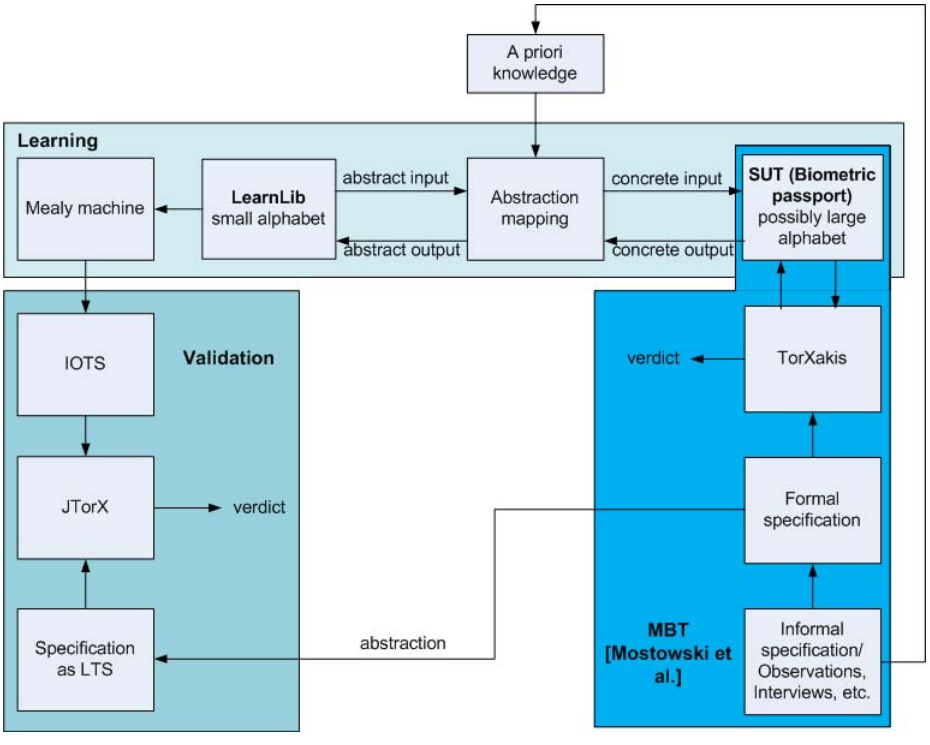


Fig. 1. Overview

specifications, observing the behavior of the *SUT*, interviews with experts, etc. Eventually, the learning algorithm generates a Mealy machine model of the *SUT*. If a reference model is available, we can validate the learned implementation to check whether it is correct with respect to the specification. In our approach, we use the testing relation **ioco** [16,17], which is implemented in the JTorX tool [5]. The Mealy machine model has to be transformed to an Input-Output Transition System (IOTS) to allow comparison with the specification represented as a LTS, see Validation box in Figure 1. We use an abstracted version of the specification to conform to the alphabet defined in the IOTS. The abstract LTS is based on a formal model created by Mostowski et al. [13] to adopt model-based testing. Their model was fed to the testing tool TorXakis (based on TorX [18]) that automatically generates and executes test cases on-the-fly. By comparing the responses of the *SUT* to those specified in the model, a verdict can be made, see MBT box in Figure 1.

3 Inference and Abstraction of Mealy Machines

In this section, we present basic principles of Mealy machines, how to infer them and to what extent abstraction techniques can be useful within learning.

3.1 Mealy Machines

A *(nondeterministic) Mealy machine (MM)* is a tuple $\mathcal{M} = \langle I, O, Q, q^0, \rightarrow \rangle$, where

- I, O and Q are finite, nonempty sets of input symbols, output symbols, and states, respectively,
- $q^0 \in Q$ is the initial state, and
- $\rightarrow \subseteq Q \times I \times O \times Q$ is the transition relation.

We write $q \xrightarrow{i/o} q'$ if $(q, i, o, q') \in \rightarrow$, and $q \xrightarrow{i/o}$ if there exists a q' such that $q \xrightarrow{i/o} q'$. Mealy machines are assumed to be *input enabled*: for each state q and input i, there exists an output o such that $q \xrightarrow{i/o}$. The transition relation is extended to sequences by defining $\overset{u/s}{\Rightarrow}$ to be the least relation that satisfies, for $q, q', q'' \in Q$, $u \in I^*$, $s \in O^*$, $i \in I$, and $o \in O$.

$$q \overset{\epsilon/\epsilon}{\Rightarrow} q$$
$$q \overset{u/s}{\Rightarrow} q' \wedge q' \xrightarrow{i/o} q'' \Rightarrow q \overset{u\,i/s\,o}{\Rightarrow} q''$$

A Mealy machine is *deterministic* if for each state q and input i there is exactly one output o and exactly one state q' such that $q \xrightarrow{i/o} q'$.

An intuitive interpretation of a Mealy machine is as follows. At any point in time, the machine is in some state $q \in Q$. It is possible to give inputs to the machine by supplying an input symbol $i \in I$. The machine then (nondeterministically) selects a transition $q \xrightarrow{i/o} q'$, produces output symbol o, and transforms itself to the new state q'.

For $q \in Q$ and $u \in I^*$, define $obs_{\mathcal{M}}(q, u)$ to be the set of output sequences that may be produced when offering input sequence u to $\mathcal{M}$, that is, $obs_{\mathcal{M}}(q, u) = \{s \in O^* \mid \exists \overline{q} : q \overset{u/s}{\Rightarrow} \overline{q}\}$. Two states $q, q' \in Q$ are *observation equivalent*, notation $q \approx q'$, if $obs_{\mathcal{M}}(q, u) = obs_{\mathcal{M}}(q', u)$, for all input strings $u \in I^*$. We write $obs_{\mathcal{M}}(u)$ as a shorthand for $obs_{\mathcal{M}}(q^0, u)$. Two Mealy machines $\mathcal{M}_1$ and $\mathcal{M}_2$ with the same sets of inputs I are *observation equivalent*, notation $\mathcal{M}_1 \approx \mathcal{M}_2$, if $obs_{\mathcal{M}_1}(u) = obs_{\mathcal{M}_2}(u)$, for all input strings $u \in I^*$. We say that $\mathcal{M}$ is *behavior deterministic* if $obs_{\mathcal{M}}(u)$ is a singleton set for each input sequence u. It is easy to see that a deterministic Mealy machine is also behavior deterministic.

Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be two Mealy machines with the same sets of input symbols. A *bisimulation* between $\mathcal{M}_1$ and $\mathcal{M}_2$ is a relation $S \subseteq Q_1 \times Q_2$ satisfying:

$$q_1\ S\ q_2 \wedge q_1 \xrightarrow{i/o}_1 q_1' \Rightarrow \exists q_2' : q_2 \xrightarrow{i/o}_2 q_2' \wedge q_1'\ S\ q_2',$$
$$q_1\ S\ q_2 \wedge q_2 \xrightarrow{i/o}_2 q_2' \Rightarrow \exists q_1' : q_1 \xrightarrow{i/o}_2 q_1' \wedge q_1'\ S\ q_2'.$$

We say that Mealy machines $\mathcal{M}_1$ and $\mathcal{M}_2$ are *bisimilar*, notation $\mathcal{M}_1 \simeq \mathcal{M}_2$, if there exists a bisimulation relation between them that contains the pair (q_1^0, q_2^0). Since the union of bisimulations is again a bisimulation, there exists a largest bisimulation. We write $q_1 \simeq q_2$ if the pair $(q_1, q_2) \in Q_1 \times Q_2$ is contained in the largest bisimulation. The following lemma is well-known and easy to prove.

Lemma 1. *Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be Mealy machines with the same sets of inputs I and let $\mathcal{M}_2$ be deterministic. Then, for $q_1 \in Q_1$ and $q_2 \in Q_2$, $q_1 \simeq q_2$ iff $q_1 \approx q_2$.*

3.2 Inference of Mealy Machines

In this section, we present the setting for inference of Mealy machines. For this purpose we make use of an extension to Angluin's L^* algorithm [4] due to Niese [14]. There is a *Teacher*, who knows a behavior deterministic Mealy machine $\mathcal{M}$, and a *Learner*, who initially has no knowledge about $\mathcal{M}$, except for its sets I and O of input and output symbols. The *Learner* can ask two types of queries to the *Teacher*:

- A *membership query* consists in asking what the response is to an input string $u \in I^*$. The *Teacher* answers with an output string $s \in O^*$.[1]
- An *equivalence query* consists in asking whether a hypothesized machine $\mathcal{H}$ is correct, i.e., whether $\mathcal{H}$ is observation equivalent to $\mathcal{M}$. The *Teacher* will answer *yes* if $\mathcal{H}$ is correct or else supply a *counterexample*, which is a string $u \in I^*$ such that u produces a different output string for both automata, i.e., $obs_{\mathcal{M}}(u) \neq obs_{\mathcal{H}}(u)$

The typical behavior of a *Learner* is to start by asking a sequence of membership queries until a "stable" hypothesis $\mathcal{H}$ can be built from the answers. After that an equivalence query is made to find out whether $\mathcal{H}$ is equivalent to $\mathcal{M}$. If the result is successful, the *Learner* has succeeded. Otherwise the returned counterexample is used to perform subsequent membership queries until converging to a new hypothesized automaton, which is supplied in an equivalence query, etc.

3.3 Inference Using Abstraction

Existing implementations of inference algorithms only proved effective when applied to machines with small alphabets (sets of input and output symbols). Practical systems, however, typically do not have small alphabets. An example are communication protocols that interact with each other via messages consisting of an action type and a number of parameters, each of which can potentially take on a large number of values. As a result, the number of input and output symbols may be astronomical. In previous work, we developed a technique for using regular inference to infer models of large-state Mealy machines [1,2]. The main idea is to transform the interface of the *SUT* by an abstraction mapping. We have adapted ideas from predicate abstraction [12,8], which has been successful for extending finite-state model checking to larger and even infinite state spaces.

Assume that we are given the task of inferring a (possibly large) Mealy machine $\mathcal{M}$ that describes the behavior of a *SUT*. In order to make the learning

[1] Actually, the term *membership query* does not conform to this setting, because we do not check whether a certain string belongs to the language or not. In fact, the term *output query* would be more appropriate. However, because it is commonly used, we decided to keep the term *membership query* in the continuation of this paper.

task feasible, we place a transducer in between the *Learner* and the *SUT*, which transforms large (parameter) domains of input and output strings of machine $\mathcal{M}$ into small ones. The combined behavior of the *SUT* and the transducer can be described by a Mealy machine $\mathcal{M}^{\mathcal{A}}$, which has smaller alphabets and (hopefully) also a smaller state space. This makes the task for the *Learner* simpler. All membership and equivalence queries generated by the *Learner* are translated by the transducer into realistic messages with possibly large (parameter) domains to accomplish the communication with the *SUT*, see the Learning part in Figure 1. Answers to queries are handled the opposite way around. To create an abstraction mapping for the transducer, we need to define when and how a concrete symbol is mapped to an abstract symbol. In general, this may require a set V of state variables to remember previous values and expressions that define when to update them. When not enough *a priori* knowledge about the behavior of $\mathcal{M}$ is available in the informal specification, an initial observation phase or interviews may be needed to create the abstraction mapping. Once we have inferred a Mealy machine equivalent to $\mathcal{M}^{\mathcal{A}}$, we have learned quite a bit about the behavior of the *SUT*, that is, about $\mathcal{M}$. In fact, in some cases (and this includes the biometric passport) we can even infer $\mathcal{M}$ from $\mathcal{M}^{\mathcal{A}}$ if we are willing to make certain structural assumptions (e.g. symmetries) about the behavior of the *SUT*.

Example. Consider a parameter Id that is used for a session establishment. We know that the *SUT* establishes a connection for the first generated Id value. All other newly generated values are treated in the same way, i.e. they are rejected. Accordingly, we can divide the values into two equivalence classes *FIRST* and *NEW*. The learning algorithm generates one of these two abstract values, which are translated to a concrete value by the abstraction mapping. Moreover, we store the concrete value in a state variable *firstId* or *newId*. These variables can be used to map a concrete output value to an abstract one. Assume that according to the specification, the *SUT* should return the Id received in the previous input message. By comparing the generated concrete output value to the value in the state variable, we can assign the output symbol one of the abstract values: $EQUAL$ or NOT_EQUAL.

4 Biometric Passport

The biometric passport is an electronic passport provided with a computer chip and antenna to authenticate the identity of travelers. The data stored on the passport are highly confidential, e.g. they might contain fingerprints or an iris scan of its owner, and are protected via several mechanisms to avoid and detect attacks. Examples of used protocols are Basic Access Control (BAC), Active Authentication (AA), and Extended Access Control (EAC) [6]. Official standards are documented in the International Civil Aviation Organisation's (ICAO) Doc 9303 [10].

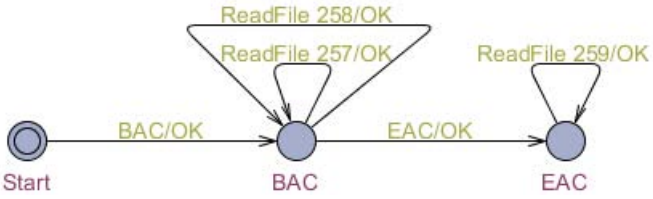


Fig. 2. Simplified model of the biometric passport

In this paper, we take a look at the interaction of the following messages:

- *Reset* resets the system.
- *GetChallenge* followed by *CompleteBAC* forms a *BAC*, which establishes secure messaging with the passport by encrypting transmitted information.
- *FailBAC* constitutes an invalid *BAC*.
- *ReadFile(int file)* tries to access highly sensitive data specified in a certain file, which is represented as an integer value in the range from 256 up to (and including) 511.
- *AA* prevents cloning of passport chips.
- *CA* followed by *TA* forms an *EAC*, which uses mutual authentication and stronger encryption than *BAC* to control access to highly confidential data.
- *FailEAC* constitutes a valid *CA* and an invalid *TA*.

For each of these messages a value *OK* or *NOK* may be returned by the *SUT*. A global overview of the valid behavior is depicted in Figure 2, where a *BAC* consists of a *GetChallenge* followed by a *CompleteBAC* and an *EAC* constitutes a *CA* followed by a *TA*. The files 257 and 258 should be readable after a BAC. File 257 contains Machine Readable Zone (MRZ) data, i.e. name, date of birth, nationality, document number, etc. whereas file 258 contains a facial image. File 259 comprises biometric data like fingerprints or an iris scan, which are only readable after a *BAC* followed by an *EAC*. All other files should not be readable at any point in time.

5 Experiments

We have implemented and applied our approach to infer a model of the biometric passport described in Section 4. In this section, we first describe our experimental setup, thereafter its application and a validation of our technique.

We used an authentic biometric passport as *SUT*. The data on the chip could be accessed via a smart card reader; JMRTD[1] served as API. We connected the *SUT* to an abstraction mapping, which performed a translation as described in Section 5.1. As *Learner*, we used the LearnLib library [15], developed at the Technical University Dortmund. This tool provides a Java implementation of the L* algorithm adapted by Niese. Because LearnLib views the *SUT* as a black box, equivalence queries can only be approximated by a large number of membership queries. In our experiments we used the W-Method by Chow [7] for equivalence approximation.

5.1 Abstraction Mapping

As described in Section 4, only the *ReadFile* message has a parameter called *file*, which can take on integer values in the range from 256 up to (and including) 511. Actually, each of these numbers has to be considered separately in the inference process, which would require a lot of time and memory space. By taking a closer look at the informal specification of the passport, we discovered that different files should be treated in the same way by the *SUT*. As one can see in Figure 2, files 257 and 258 should be readable after a BAC, 259 after a BAC followed by an EAC and the rest of the files should never be readable. Using this *a priori* knowledge about the passport, we can divide the values into three disjoint equivalence classes, which are:

- *ValidAfterBAC* refers to the files that can be read after a BAC, i.e. 257 and 258.
- *ValidAfterEAC* refers to the files that only can be read after a BAC followed by an EAC, i.e. 259.
- *NotValid* refers to the files that can never be read, i.e. all files except for 257, 258 and 259.

In the abstraction mapping an abstract value is translated to a concrete one by randomly choosing an element within the corresponding equivalence class. If the numbers are partitioned incorrectly, then there are two values in the same class that will produce a different response. This non-deterministic behavior will be detected by LearnLib, which will give an error message.

5.2 Results

The inference performed by LearnLib needed about one thousand membership queries and one equivalence query, and resulted in a model $\mathcal{H}^{\mathcal{A}}$ with five states and 55 transitions. Without our abstraction mapping, the Mealy machine would have had 1320 transitions, but also five states. The total learning time took less than one hour[2]. This is significantly shorter than deriving the model manually from the informal specs, which took about 5 hours. All results are summarized in Table 1. For presentation purposes, we have depicted the model as follows: (1) we removed self-transitions with *NOK* as output. Because the model is input-enabled all missing entries refer to this kind of transition. (2) Transitions with same source location, output symbol and next location (but with different input symbols) are merged by concatenating the input symbols, separated by a bar (|). The resulting transition diagram has five locations and 19 transitions as shown in Figure 3.

The implementation of the biometric passport does not respond to a *Reset* input. For all other outputs the reaction time is dependent on the input symbol. If the waiting time for an output is too short, then an output symbol may be

[2] The experiments have been carried out on a PC with an Intel Pentium M 1.86GHz processor and 1GB of RAM.

Table 1. Learning statistics

Membership queries	1078[a]
Total input symbols used in membership queries	4158
Average membership query length	3,867
Equivalence queries	1
Total learning time	< 60 minutes

[a] This number does not include membership queries used for equivalence approximation.

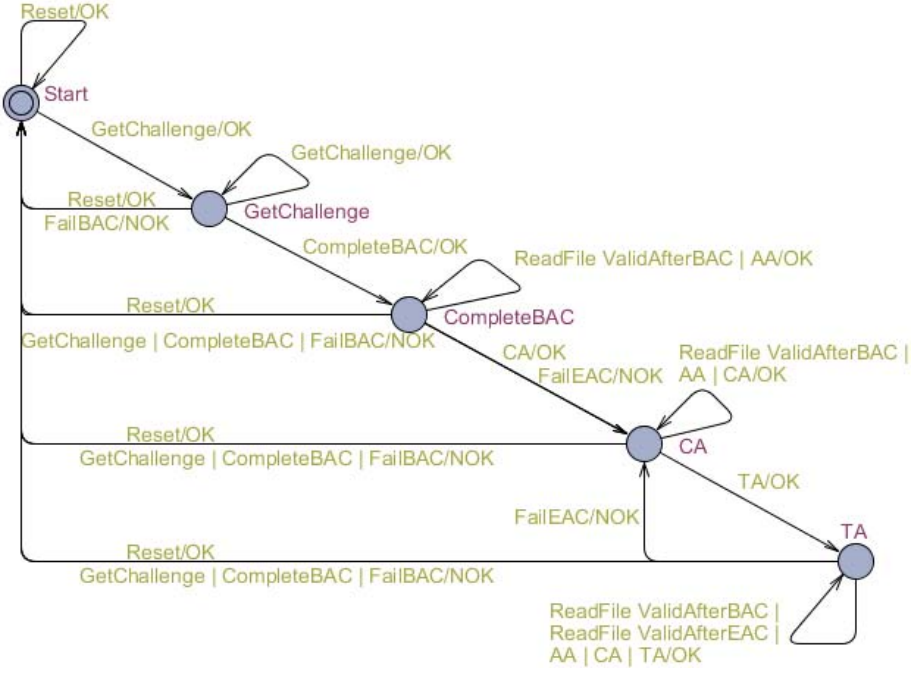


Fig. 3. Learned model $\mathcal{H}^{\mathcal{A}}$ of the biometric passport

returned after a *timeout* has been assumed. In contrast, if the waiting time is too long, then the passport application crashes after certain inputs. As a solution, we changed the API of the *SUT*, so that it returns an *OK* symbol for each *Reset* input. By always returning an output symbol, we do not have to struggle with appropriate waiting times per input symbol. Instead, we wait until an output is received.

According to the passport specification, the implementation should be deterministic. However, surprisingly, the passport application sometimes exhibits non-deterministic behavior. LearnLib is restricted to infer behavior deterministic Mealy machines and cannot cope with non-deterministic behavior. Analyzing the external behavior of the system revealed that after a *GetChallenge*,

CompleteBAC, *CA*, *TA* input sequence mostly an *OK* is returned, but in some rare cases it can also be a *NOK*. Together with Mostowski et al. we tried to examine the internal behavior of the application to understand where the nondeterminism originates from. During their work this problem has also been encountered, but it has never been reported. Because a *TA* call includes numerous complex and long calculations, a problem can arise at several places. Moreover, external circumstances may influence the produced results like connection to or temperature of the smart card reader. In the end, we could not clearly determine the fault location and had to accept that the inference can fail once in a while.

5.3 The Behavior of the SUT

We assume that the behavior of the digital passport can be modeled in terms of a behavior deterministic Mealy machine $\mathcal{M}$. Clearly, due to the abstraction that we applied, the learned model $\mathcal{H}^{\mathcal{A}}$ is not equivalent to $\mathcal{M}$: even the alphabets are different. Let $\mathcal{M}^{\mathcal{A}}$ be the Mealy machine obtained from $\mathcal{M}$ by renaming each action *ReadFile*(*file*) in accordance with the abstraction mapping defined in Section 5.1. We assume that also $\mathcal{M}^{\mathcal{A}}$ is behavior deterministic. Since the SUT and the transducer together behave like $\mathcal{M}^{\mathcal{A}}$, the learned model $\mathcal{H}^{\mathcal{A}}$ should be equivalent to $\mathcal{M}^{\mathcal{A}}$. LearnLib implements several algorithms that can be used to "approximate" equivalence queries, that is, to establish that the hypothesized machine $\mathcal{H}^{\mathcal{A}}$ is observation equivalent to the model $\mathcal{M}^{\mathcal{A}}$ of the teacher. We have used the well-known W-method of [7] (see also [11]). This method assumes a known upper bound on the number of states n of $\mathcal{M}^{\mathcal{A}}$. Depending on n the W-method provides a test sequence of input symbols u with the property that $\mathcal{M}^{\mathcal{A}}$ and $\mathcal{H}^{\mathcal{A}}$ are observation equivalent iff they produce the same output in response to u. But assuming that we have established equivalence of $\mathcal{M}^{\mathcal{A}}$ and $\mathcal{H}^{\mathcal{A}}$, what have we learned about $\mathcal{M}$?

We reverse the abstraction mapping and construct a "concrete" model $\mathcal{H}$ of the passport as follows. We replace each *ValidAfterBAC* transition in $\mathcal{H}^{\mathcal{A}}$ by two transitions with the same source and target but with labels *ReadFile(257)* and *ReadFile(258)*, respectively. Similarly, we replace each transition with label *ValidAfterEAC* by an identical transition with label *ReadFile(259)*. Finally, we replace each transition with label *NotValid* by 253 identical transitions with labels *ReadFile(256)*, *ReadFile(260)*, *ReadFile(261)*, ..., *ReadFile(511)*, respectively.

The following theorem states that if $\mathcal{M}$ treats equivalent input symbols in an equivalent way, observation equivalence of $\mathcal{M}^{\mathcal{A}}$ and $\mathcal{H}^{\mathcal{A}}$ implies observation equivalence of $\mathcal{M}$ and $\mathcal{H}$. So provided we are willing to make a structural assumption about the behavior of the biometric passport, our abstraction does not lead to any loss of information.

Theorem 1. *Call two input actions i_1, i_2 of Mealy machine $\mathcal{M}$* equivalent*, notation $i_1 \equiv i_2$, if they are mapped to the same abstract action. Suppose that in $\mathcal{M}$ equivalent inputs induce identical outputs and equivalent successor states:*

$$i_1 \equiv i_2 \wedge q \xrightarrow{i_1/o_1} q_1 \wedge q \xrightarrow{i_2/o_2} q_2 \Rightarrow o_1 = o_2 \wedge q_1 \approx q_2.$$

Then $\mathcal{M}^{\mathcal{A}} \approx \mathcal{H}^{\mathcal{A}}$ implies $\mathcal{M} \approx \mathcal{H}$.

Proof. Suppose $\mathcal{M}^{\mathcal{A}} \approx \mathcal{H}^{\mathcal{A}}$. We must show that $\mathcal{M} \approx \mathcal{H}$. Since $\mathcal{H}^{\mathcal{A}}$ is deterministic, Lemma 1 implies that $\mathcal{M}^{\mathcal{A}} \simeq \mathcal{H}^{\mathcal{A}}$. Let S be the maximal bisimulation between $\mathcal{M}^{\mathcal{A}}$ and $\mathcal{H}^{\mathcal{A}}$. It suffices to prove that S is a bisimulation between $\mathcal{M}$ and $\mathcal{H}$, since by Lemma 1 this implies $\mathcal{M} \approx \mathcal{H}$.

Since S relates the initial states of $\mathcal{M}^{\mathcal{A}}$ and $\mathcal{H}^{\mathcal{A}}$, it also relates the initial states of $\mathcal{M}$ and $\mathcal{H}$.

Suppose that $(q, r) \in S$ and $q \xrightarrow{i/o}_{\mathcal{M}} q'$. Suppose that the abstraction function maps i to j. Then $q \xrightarrow{j/o}_{\mathcal{M}^{\mathcal{A}}} q'$. Since S is a bisimulation between $\mathcal{M}^{\mathcal{A}}$ and $\mathcal{H}^{\mathcal{A}}$, there exists a state r' such that $r \xrightarrow{j/o}_{\mathcal{H}^{\mathcal{A}}} r'$ and $(q', r') \in S$. By construction of $\mathcal{H}$, $r \xrightarrow{i/o}_{\mathcal{H}} r'$.

Now suppose that $(q, r) \in S$ and $r \xrightarrow{i/o}_{\mathcal{H}} r'$. Suppose the abstraction function maps i to j. Then, by construction of $\mathcal{H}$, $r \xrightarrow{j/o}_{\mathcal{H}^{\mathcal{A}}} r'$. Since S is a bisimulation between $\mathcal{M}^{\mathcal{A}}$ and $\mathcal{H}^{\mathcal{A}}$, there exists a state q' such that $q \xrightarrow{j/o}_{\mathcal{M}^{\mathcal{A}}} q'$ and $(q', r') \in S$. Therefore, by construction of $\mathcal{M}^{\mathcal{A}}$, there exists a concrete input label k such that the abstraction maps k to j and $q \xrightarrow{k/o}_{\mathcal{M}} q'$. Since $\mathcal{M}$ is input enabled, there exists an output label p and a state q'' such that $q \xrightarrow{i/p}_{\mathcal{M}} q''$. Oberve that $i \equiv k$. Hence, by the assumption of the theorem, $o = p$ and $q' \approx_{\mathcal{M}} q''$. By construction of $\mathcal{M}^{\mathcal{A}}$, this implies $q' \approx_{\mathcal{M}^{\mathcal{A}}} q''$ By Lemma 1, applied to $\mathcal{M}^{\mathcal{A}}$ and $\mathcal{H}^{\mathcal{A}}$, using the fact that $(q', r') \in S$, we obtain $(q'', r') \in S$.

5.4 Validation

To validate the learned model of the biometric passport, we compared it to a reference model taken from Mostowski et al. [13]. The specification is a LTS made in Haskell[3] and has to be transformed to a different format to allow comparison with the inferred Mealy machine described in the *DOT*[4] language.

For the comparison, we used JTorX [5], a tool to test whether the **ioco** testing relation holds between a given specification and a given implementation. Intuitively, an implementation $i \in \mathcal{IOTS}(L_I, L_U)$ is input-output conforming to specification $s \in \mathcal{LTS}(L_I, L_U)$ if any experiment derived from s and executed on i leads to an output from i that is foreseen by s. For a formal definition, we refer to [17]. We have supplied JTorX with the specification and implementation as LTS - represented in Aldebaran[5] format. The learned Mealy machine has been transformed to a LTS by splitting each transition into two with the input symbol on the first transition and the output on the second one connected by an additional state. As a result, the input-enabledness of the Mealy machine gets lost. To convert the learned LTS to an IOTS, JTorX adds self-loop transitions to the according states. Furthermore, we removed the output *OK* for a *Reset* input, because it is unknown by the specification, see Section 5.2.

[3] http://www.haskell.org
[4] http://www.graphviz.org
[5] http://www.inrialpes.fr/vasy/cadp/man/aldebaran.html

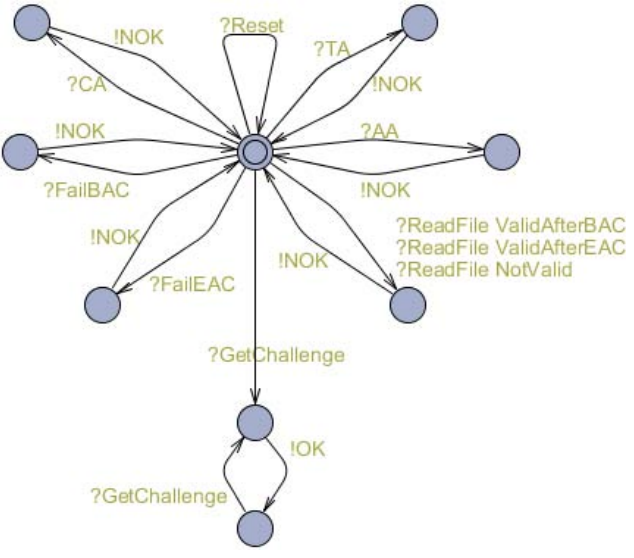


Fig. 4. Specification fragment of the biometric passport

According to JTorX, the implementation is **ioco** conforming to the specification, but not vice versa. This is not surprising as the learned model is input-enabled while the specification is not. For example, the specification does not specify a *CompleteBAC* input in the initial state while the learned implementation does, see Figure 4 for a fragment of the specification. We only show the inputs in the initial state with according outputs, because the entire model contains too many states and transitions. As one can see, the automaton corresponds to a Mealy machine. Except for the *Reset* input, each input is followed by an output. If we would transform the specification to a Mealy machine, it would not be input-enabled. Because LearnLib infers an input-enabled Mealy machine of the implementation, it contains more behavior than described by the specification, which is allowed by the **ioco** testing relation.

6 Conclusions and Future Work

Using regular inference and abstraction, we have managed to infer a model of the biometric passport that describes how the passport responds to certain input sequences. Although quite a number of papers have been written on regular inference of state machines, the number of real applications to reactive systems is still limited. The case study that we describe here is a small but real application. The new biometric passport is going to be used by millions of people, and it is vital that the confidential information stored on this passport is well-protected. Our model, which slightly refines the earlier model of [13], may serve as a reference model for testing different implementations of the biometric passport.

The data abstraction that we applied when learning the passport may seem rather obvious (and indeed is much simpler than the abstractions applied in [1,2]), but is nevertheless crucial for the successful application of our learning framework. In order to prevent brute force attacks, the biometric passport only allows for about one input message per second. Without abstraction, the time needed to apply the framework (and in particular the approximation of equivalence queries via e.g. the W-method) would become prohibitively large. We have proven that under some reasonable assumptions about the behavior of the biometric passport, our abstraction does not lead to any loss of information.

The earlier model of [13] has been created manually in about 5 hours, whereas our model has been produced automatically in less than one hour. Our ambition is to further develop the learning framework, so that also for other applications it becomes feasible to mechanize and speed-up the time-consuming and error prone process of constructing reference models.

Due to the problems with the non-deterministic behavior of the passport, an obvious topic for future research is to extend our approach to inference of non-deterministic systems. Such an extension will be essential, when doing more real-world case studies like this one.

If inferring an input-enabled Mealy machine is too time-consuming and we are only interested in parts of the implementation, we may extend our abstraction mappings with an *interface automaton (IA)* as suggested by [3]. An interface automaton [9] is a labelled transition system with input and outputs, where certain input actions may be illegal in certain states. When an input symbol or sequence generated by the learning algorithm is not allowed by the specified *IA*, this part of the implementation will not be inferred. By adding restrictions, we can focus on those parts of the implementation that are described by the specification.

Acknowledgement. We are grateful to Falk Howar from the TU Dortmund for his generous LearnLib support, and Wojciech Mostowski for providing assistance with JMRTD.

References

1. Aarts, F.: Inference and Abstraction of Communication Protocols. Master's thesis, Radboud University Nijmegen and Uppsala University (2009)
2. Aarts, F., Jonsson, B., Uijen, J.: Generating Models of Infinite-State Communication Protocols using Regular Inference with Abstraction. In: Proceedings ICTSS 2010, 22nd IFIP International Conference on Testing Software and Systems (2010)
3. Aarts, F., Vaandrager, F.: Learning I/O Automata. In: Proceedings CONCUR 2010, 21th International Conference on Concurrency Theory, pp. 71–85 (2010)
4. Angluin, D.: Learning regular sets from queries and counterexamples. Information and Computation 75(2), 87–106 (1987)
5. Belinfante, A.: JTorX: A tool for on-line model-driven test derivation and execution. In: Esparza, J., Majumdar, R. (eds.) TACAS 2010. LNCS, vol. 6015, pp. 266–270. Springer, Heidelberg (2010)

6. BSI. Advanced security mechanisms for machine readable travel documents - extended access control (eac) - version 1.11. Technical Report TR-03110, German Federal Office for Information Security (BSI), Bonn, Germany (2008)
7. Chow, T.S.: Testing software design modeled by Infinite-state machines. IEEE Trans.on Software Engineering 4(3), 178–187 (1978); Special collection based on COMPSAC
8. Clarke, E.M., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample- guided abstraction refinement for symbolic model checking. Journal of the ACM 50(5), 752–794 (2003)
9. de Alfaro, L., Henzinger, T.A.: Interface automata. In: Gruhn, V. (ed.) Proceedings of the Joint 8th European Software Engineering Conference and 9th ACM SIGSOFT Symposium on the Foundation of Software Engineering (ESEC/FSE-2001). Software Engineering Notes, vol. 26, pp. 109–120. ACM Press, New York (September 2001)
10. ICAO. Doc 9303 - machine readable travel documents - part 1-2. Technical report, International Civil Aviation Organization, 6th edn. (2006)
11. Lee, D., Yannakakis, M.: Principles and methods of testing Finite state machines a survey. Proc. IEEE 84(8), 1090–1126 (1996)
12. Loiseaux, C., Graf, S., Sifakis, J., Boujjani, A., Bensalem, S.: Property preserving abstractions for the verifIcation of concurrent systems. Formal Methods in System Design 6(1), 11–44 (1995)
13. Mostowski, W., Poll, E., Schmaltz, J., Tretmans, J., Wichers Schreur, R.: Model-based testing of electronic passports. In: Cofer, D., Fantechi, A. (eds.) FMICS 2008. LNCS, vol. 5596, pp. 207–209. Springer, Heidelberg (2009)
14. Niese, O.: An integrated approach to testing complex systems. Technical report, Dortmund University, Doctoral thesis (2003)
15. Raffelt, H., Steffen, B., Berg, T.: Learnlib: a library for automata learning and experimentation. In: FMICS 2005: Proceedings of the 10th International Workshop on Formal Methods for Industrial Critical Systems, pp. 62–71. ACM Press, New York (2005)
16. Tretmans, J.: Test generation with inputs, outputs and repetitive quiescence. Software - Concepts and Tools 17(3), 103–120 (1996)
17. Tretmans, J.: Model based testing with labelled transition systems. In: Hierons, R.M., Bowen, J.P., Harman, M. (eds.) FORTEST. LNCS, vol. 4949, pp. 1–38. Springer, Heidelberg (2008)
18. Tretmans, J., Brinksma, H.: TorX: Automated model-based testing. In: Hartman, A., Dussa-Ziegler, K. (eds.) First European Conference on Model-Driven Software Engineering, pp. 31–43 (December 2003)

From ZULU to RERS

Lessons Learned in the ZULU Challenge*

Falk Howar, Bernhard Steffen, and Maik Merten

University of Dortmund, Chair of Programming Systems,
Otto-Hahn-Str. 14, 44227 Dortmund, Germany,
Tel.: ++49-231-755-7759; Fax: ++49-231-755-5802
{falk.howar,steffen,maik.merten}@cs.tu-dortmund.de

Abstract. This paper summarizes our experience with the ZULU challenge on active learning without equivalence queries, presents our winning solution, investigates the character of ZULU's rating approach, and discusses how this approach can be taken further to establish a framework for the systematic investigation of domain-specific, scalable learning solutions for practically relevant application. In particular, it discusses the RERS initiative, which provides a community platform together with a learning framework that allows users to interactively compose complex learning solutions on the basis of libraries for various learning components, system connectors, and other auxiliary functionality. This framework will be the backbone for an extended challenge on learning in 2011.

1 Motivation

In the last decade, active automata learning, an originally merely theoretical enterprise, got attention as a method for dealing with black-box or third party systems. Applications ranged from the support of formal verification, e.g. for assume guarantee reasoning [10,22], to usage of learned models as the basis for regression testing. In the meantime there exist a number of approaches exploiting active learning for validation [23,28,14,15,5,1,6]. This success may seem surprising, because automata learning, in practice, is inherently neither correct nor complete, and there are only few examples of learned systems of significant size. Hardly any have more than 1,000 states, and to our knowledge the largest learned realistic system is a router with about 22,000 states [24]. On the other hand, there does not seem to be a good alternative for dealing with black-box systems.

This situation calls for a concerted action to improve the state of the art of practical learning. A first attempt in this direction was the ZULU challenge. ZULU asked for learning solutions that function without the use of so-called equivalence queries, which classically could be used to automatically derive a counterexample distinguishing a learned hypothesis model from the target system until equivalence has been established. The reason for excluding equivalence

* This work is supported by the European FP 7 project CONNECT (IST 231167).

T. Margaria and B. Steffen (Eds.): ISoLA 2010, Part I, LNCS 6415, pp. 687–704, 2010.

queries, which guaranteed the correctness and completeness of the theoretical framework, was their lack of realism: in essentially all practical applications, equivalence queries need to be approximated by the so-called membership queries, which often can be implemented via testing. ZULU therefore took exactly this approach and allowed membership queries only, and, indeed, only a comparatively small number of them. Also this latter limitation was motivated by a practical perspective: typically, the approximations of equivalence queries consume an enormous amount of membership queries, and this was considered unrealistic as well.

This paper summarizes our experience with the ZULU challenge on active learning without equivalence queries, presents our winning solution, and

- investigates the character of ZULU's rating approach, which ranks solutions according to their prediction quality for language containment based on a ZULU-generated test suite, and
- discusses how this approach can be taken further to establish a framework for the systematic investigation of domain-specific, scalable learning solutions for practically relevant application.

In particular, we discuss the RERS initiative [8], which provides a community platform together with a learning framework that allows users to interactively compose complex learning solutions on the basis of libraries for various learning components, system connectors, and other auxiliary functionality.

With RERS we want to establish a community of researchers and practitioners interested in the practical application of automata learning technology. RERS stands for Regular Extrapolation of Reactive Systems. We chose the term Regular Extrapolation to indicate that in practice we will not be able to fully infer the behavior of the target system, which often will not be regular anyway. Rather we are able to construct in some sense optimal regular images/views of these behaviors. We are convinced that a concerted effort for improving these techniques will lead to industrial scale learning solutions in near future.

Outline: After recapitulating briefly the central idea of active learning in Section 2, we will discuss the ZULU challenge, our approach to it, the outcome, and further observations in Section 3. Subsequently, we will give a short introduction to the RERS initiative in Section 4, and conclude in Section 5.

2 Active Learning

Active learning (or *query learning*) attempts to construct a deterministic finite representation, e.g., a deterministic finite automaton (DFA), that matches the behavior (language) of a given target system on the basis of observations of the target system and perhaps some further information on its internal structure. It poses *membership queries* that test whether certain strings/words (potential runs) are contained in the target system's language (its set of runs), and *equivalence queries* that compare intermediately constructed hypothesis automata for

equivalence with the target system. As long as the equivalence queries fail, they produce counterexamples revealing some of the remaining differences between hypothesis and target system. These counterexamples are then taken as the basis for further refinement of the hypothesis. Learning terminates successfully as soon as an equivalence query signals success.

Angluin's algorithm L^*, in its basic form, starts with a hypothesis automaton with only one state and refines this automaton on the basis of query results. In scenarios like for ZULU, which (like most practical settings) do not provide an equivalence oracle for reliably answering equivalence queries (see below), three main steps are iterated:

1. *refining the hypothesis* by means of membership queries,
2. *checking for equivalence* on the basis of membership queries e.g., following ideas from *conformance testing* (e.g., [9,13]), and
3. *analyzing counterexamples* as a means to initiate the next refinement step.

The procedure terminates (for finite state target systems) with a state-minimal deterministic (hypothesis) automaton equivalent to the target system. This is the consequence of the following dual characterizations of states, resembling the idea of Nerode's congruence [21,20]:

from below: by a set, $S \subset \Sigma^*$, of *access sequences*. This characterization of state is too fine, as different words $s_1, s_2 \in S$ may lead to the same state in the target system. L^* constructs such a set S, containing access sequences to all states of the target automaton. In addition, it maintains a second set, SA, which together with S covers all transitions of the hypothesis automaton. During learning, the following invariant is repetitively established: $SA = (S \cdot \Sigma) \setminus S$. L^* initializes S with $\{\varepsilon\}$, the set containing only the access sequence to the initial state, and, accordingly, SA with Σ, covering all transitions leaving in the initial state.

from above: by ordered set(s), $D \subset \Sigma^*$, of distinguishing sequences or *futures*. These sets can be regarded as a projection of Nerode's residual languages. Starting with the singleton set $\{\varepsilon\}$, L^* successively extends D until it suffices to identify all states of the smallest deterministic automaton.

The sets S, SA, and D are successively refined during the learning process until a model is reached, which is state-minimal and equivalent (at least up to the distinctive power of the given approximation of the equivalence oracle) to the target system. Algorithmic details and correctness arguments can be found in [2,17,26,20,19,27].

3 The ZULU Competition

ZULU [11] is a competition in active learning from membership queries: contestants had to develop active learning algorithms (following the general approach due to Dana Angluin [2]). Essential to ZULU have been two main ideas:

No equivalence queries: Equivalence queries (in the theoretical formulation of active learning) compare a learned hypothesis model with the target system for language equivalence and, in case of failure, return a counterexample exposing a difference. Their realization is rather simple in simulation scenarios: if the target system is a model, equivalence can be tested explicitly. In practice, however, the system under test will typically be some kind of black-box and equivalence queries will have to be simulated using membership queries [16]. One goal of the ZULU competition was to intensify research in this practical direction.

Limited membership queries: Equivalence queries can be simulated using model-based testing methods [7] (The close relation between learning and conformance testing is discussed in [4]). If, e.g., an upper bound is known on the number of states the target system can have, the W-method [9] or the Wp-method [13] can be applied. Both methods have an exponential complexity (in the size of the target system and measured in the number of membership queries needed).

By allowing only a (very) limited number of membership queries for every problem, the ZULU competition forced the contestants to change the objective from 'trying to prove equivalence', e.g., by using conformance testing techniques, to 'finding counterexamples fast'.

These two ideas led to the following concrete competition scenario. The contestants had to compete in producing models of finite state acceptors by means of membership queries, which the ZULU platform would answer. For every task, the number of granted membership queries was determined by the ZULU system as the number of membership queries needed by some basic Angluin-style reference implementation to achieve a prediction quality of over 70%. Solutions were ranked according to their quality of prediction relative to a ZULU-generated suite of 1800 test words (the same suite as used to determine the number of allowed membership queries).

During the competition, all contestants had to compete in 12 categories: 9 'regular' categories ranging over finite state acceptors with (a) growing numbers of states and (b) growing alphabets, as well as 3 'extra' categories, in which the number of granted membership queries was reduced further. The overall winner was determined as the contestant with the best average ranking. A detailed description of the technical details can be found on the ZULU web page [12].

The background of our team at the TU Dortmund is the development of realistic reliable systems [24,25]. Accordingly, we are working on techniques to make active learning applicable to industrial size systems. Of course this also requires to (1) saving membership queries and (2) finding counterexamples fast, but for man-made systems. It turned out that dealing with randomly generated systems changes the situation quite a bit. Our solutions therefore arose in a two step fashion:

1. a generically optimized basis, not exploiting any knowledge about the structure or origin of the considered target systems. The main strategy here was

to follow an *evolutionary* approach to hypothesis construction which explicitly exploits that the series of hypotheses are successive refinements. This helps organizing the membership and equivalence queries.
2. a subsequent customization for exploiting the fact that we are dealing with randomly generated systems. Here we were able to exchange our traditionally breadth-first-oriented approximations of an equivalence oracle, which exploit ideas from conformance testing, by a very fast counterexample finder. Also this customization benefits from the evolutionary character of the hypothesis construction.

The careful experimental investigation of the above options and its variants profited from our learning framework, the LearnLib [25], which we recently considerably extended. In particular, it enabled us to build a highly configurable learning algorithm, which can mimic most of the known algorithms, as well as all our variations in a simple fashion. In addition, the experimentation facilities of LearnLib, comprising, e.g.,remote execution, monitoring, and statistics, saved us a lot of time. We are planning to provide contestants of the RERS challenge [8] with all these facilities in order to allow contestants a jump start.

3.1 A Configurable Inference Framework

The main algorithmic pattern we used for the ZULU competition can best be described as generalized *Observation Pack* [3]: a pack is a set of components (which usually form the observation table). In the original discussion an observation pack is introduced only as a unifying formalism for the algorithms of [2,17]. We used a combination of discrimination trees [17] and reduced

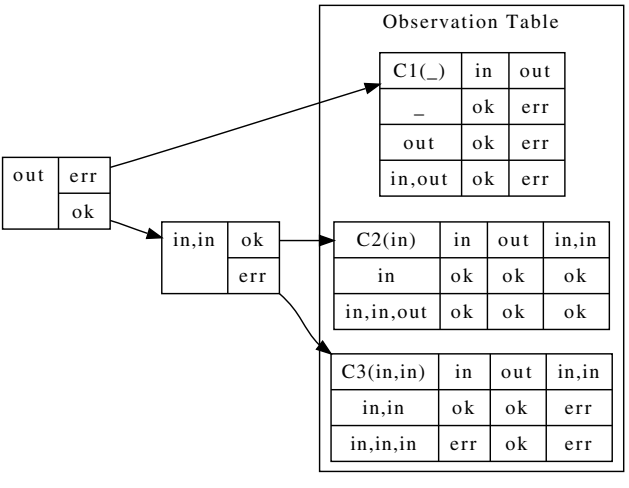


Fig. 1. Extended Observation Packs

observation tables [26] to actually implement an observation pack. Fig. 1 illustrates the resulting data structure for a Mealy machine example (a three element buffer).

The observation table is split into independent components (essentially small observation tables in their own), each representing one state of the current hypothesis model, and being defined by one access sequence from the set S (see upper left corner). The second row of each component corresponds to the usual characterizing row for this access sequence in terms of some distinguishing futures taken from D, which label the columns. Additional rows stand for words of SA which happen to possess the same characterization in terms of the considered distinguishing futures, and therefore represent the same state in the hypothesis model. Please note that the description above does not hold for the third component which still needs to be split according to its difference with respect to the distinguishing future *in* (see below).

The left part of Fig. 1 shows the corresponding discrimination tree. Its nodes are labeled with elements of D, and its edges with the corresponding outcome of a membership query (here `ok` and `err` - one could also have chosen accept, non-accept). In the considered situation the discrimination tree simply reflects the fact that the first component can be separated from the other two due to the outcome concerning the distinguishing future `out`, whereas the second and third component can be separated from each other due to the outcome concerning the distinguishing future `in,in`. As indicated already, the third component could be split further due to the outcome concerning the distinguishing future `in`. Such a situation can only arise in the Mealy case, where the set of distinguishing futures is initialized with the full alphabet. In this concrete case, the alphabet symbol `in` is contained in the initial set of distinguishing futures but not yet in the discrimination tree.

In order to enter a new access string s into the discrimination tree, one starts with the root of the discrimination tree and successively poses the membership queries $s \cdot f$, where f is the distinguishing future of the currently considered node, and continues the path according to the corresponding output. If a corresponding continuation is missing, the discrimination tree is extended by establishing a new component for the considered access string. Finally, splitting a component requires the introduction of a new node in the discrimination tree, like e.g. for the distinguishing future `in` to split the third component.

This realization allows us to easily switch between different strategies for handling counterexamples (e.g., [26,19,27]), as well as to use non-uniform observation tables, i.e., observation tables where for different access sequences different distinguishing futures may be considered.

To enable working with non-uniform observation tables, we extended the strategy for analyzing counterexamples from [26]. In its original form, this strategy produces a new distinguishing future d by means of a binary search over the counterexamples. During the search prefixes of the counterexamples are replaced by access sequences to the according states in the hypothesis model, in order to maintain as much of the hypothesis model as possible. Taking this idea a bit

further allows us to guarantee that the access sequence s for a new component is always taken from SA. This automatically maintains the structure of the spanning tree, and it guarantees that only the component containing s is refined by the new distinguishing future d. The result of this refinement is a new node in the discrimination tree and two components (following the approach from [17]).

For ZULU, we configured two versions of our learning algorithm, both using this new strategy for analyzing counterexamples. The registered algorithms differed as follows.

Initial set of distinguishing futures: In one configuration, the initial set of distinguishing futures was initialized as $\{\varepsilon\}$ (as in the literature). In the other configuration, we used $\{\varepsilon\} \cup \Sigma$ in order to simulate the effect of changing from DFA to Mealy models. Please note that considering the empty word also for the Mealy configuration is a technical trick to better deal with the DFA-like systems considered here.

Observation Table: We used uniform and non-uniform observation tables. In a non-uniform table, the sets of distinguishing futures are managed independently for each component (cf. [3]).

Both decisions emphasize the same trade-off. Using the complete alphabet in the initial set and using a uniform observation table can be understood as a heuristic for finding counterexamples (in form of *unclosure*) in the table and thus reducing the number of equivalence queries. Using a non-uniform table and only proven distinguishing futures leads to using less membership queries but more equivalence queries.

3.2 Continuous Equivalence Queries

Essentially, active learning algorithms proceed in rounds triggered by negative outcomes of *equivalence queries* for successively improved hypotheses: returned counterexamples are analyzed and exploited to initialize the next round of refinement. Classically, each of these rounds were considered independent, and in fact, equivalence queries were simply provided with the current hypothesis model, which itself was independently constructed for each round. No knowledge about the algorithmic history was exploited. This is not too surprising for two reasons:

- Classically, equivalence queries were considered as atomic, with no need for a dedicated optimization, a point of view also supported by experimental settings, where the target systems are given as automata, which can efficiently be compared with the hypothesis automata.
- However, there is also a more technical argument. The hypothesis automata, which are refined during the learning process, are defined solely by the state characterization from above (their characterizing set), which is not fully reflected in the hypotheses. Thus the knowledge of the hypotheses alone is insufficient to establish the required notion of refinement.

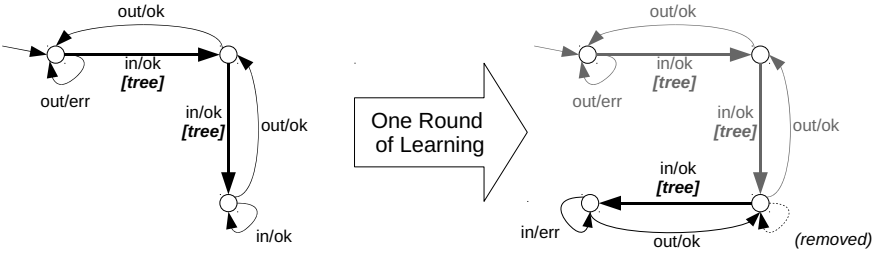


Fig. 2. Evolving Hypothesis

In the meantime it is widely accepted that membership query-based approximations are key to practical application. The ZULU challenge itself is a very strong indication of this new perspective. Moreover, there are ways to strongly link the characterizations from below and above in a way that allow some kind of *incremental hypothesis construction* by means of a global, continuous equivalence querying process. Key to this approach are Rivest's and Schapire's reduced observation tables together with their way of analyzing counterexamples [26]). In their setting, the prefix-closed set S can be understood as a successively produced spanning tree of the target automaton, whose set of nodes is extended by elements from the SA-set. Thus there is always a unique representation of the states of the hypothesis automaton in terms of the monotonically increasing, prefix-closed set S. This invariant is maintained by the specific treatment of counterexamples:

> Each counterexample is minimized by means of binary search to a word/string sa of SA, followed by a distinguishing future d that forces sa to end up in a new state.

This form of counterexample allows maintaining the above invariant by moving the corresponding s from SA to S, add d to the set of futures D, and to continue with the usual procedure establishing closedness. Besides avoiding the construction of hypotheses from scratch, this procedure leads to a sequence of incrementally refined hypotheses, which allows for organizing the equivalence tests from a global perspective, sensitive to all the tests which have been performed in earlier iterations.

Fig. 2 shows how an evolving hypothesis is refined during one round of the learning process. The hypothesis in the left of the figure corresponds to the extended observation packs from Fig. 1. All transitions are labeled by annotations, indicating whether they belong to the spanning-tree or not. This information is vital, as it indicates proven knowledge. Adding a new state to the hypothesis leaves most of the original hypothesis and its annotations unaffected. In the example of Fig. 2 only the former loop at the third state is modified and two new transitions are introduced.

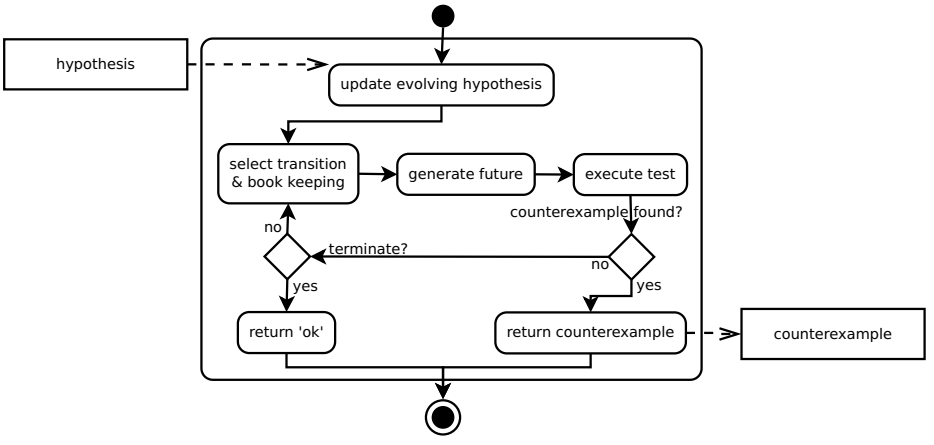


Fig. 3. Continuous Equivalence Query

Beyond being a possible target for counterexample reduction (after some oracle provided an arbitrarily structured counterexample), the counterexample pattern

$$c = sa \cdot d, \text{ with } sa \in SA \text{ and } d \in \Sigma^+.$$

turned out to be ideal for realizing membership oracle-based equivalence oracles, or better, for implementing a method for revealing counterexamples fast along the lines shown in Fig. 3. Counterexample candidates sa are are tested for some heuristically chosen futures d until either a counterexample is found or some termination criterion is met. The effectiveness of the search heuristics for selecting counter example candidates and finding distinguishing futures d may strongly depend on side knowledge. For the ZULU challenge, we exploited our knowledge that the languages were randomly generated as follows:

Select transitions & Book keeping: For the *E.H.Blocking* algorithm, transitions from the SA-set were chosen randomly. Once used, a transition was excluded from subsequent tests. When no transitions were left to choose from, all transitions were re-enabled. The *E.H.Weighted* algorithm uses weights on all transitions, which are increased each time a transition is selected, and chooses transitions with a probability inversely proportional to its weight.

Generate futures: The futures were generated randomly with increasing length. The length was initialized as some ratio of the number of states of the hypothesis automaton, and was increased after a certain number of unsuccessful tests. The exact adjustment of the future length was developed in a experimentally to fit the properties of the problems in the ZULU challenge. This strategy of guessing comparatively long futures d turned out to be rather effective. It reduced the number of required membership queries to an average of 2-4, which radically outperformed our initial breath-first trials. Of course,

this is very much due to the profile of the ZULU examples, and indeed, this choice of futures was the only truly domain-dependent optimization we developed.

We did not use an explicit termination criterion. A query terminated as soon as the number of queries granted by ZULU was exhausted.

3.3 Results

For the actual competition, we registered six candidate algorithms, split in two groups of three:

- the first group used a non-uniform observation table with a DFA-style initial set of distinguishing futures, and
- the second group used a uniform observation table with a (modified) Mealy-style initial set of distinguishing futures.

Both groups were equipped with the same three equivalence checking algorithms: (1) E.H.Blocking, (2) E.H.Weighted, and (3) plain random walks. As the random walks algorithm simply tested randomly generated words, the option for continuous equivalence queries did not apply. Table 1 shows the configuration of the algorithms, their average scores during the training phase and the eventual rankings from the competition.

Apparently, group 1 is far superior: about 10 points, and this in a setting where there are only 0.45 points between the first and the sixth place. In order understand these results better, we investigated Problem 49763507 in more detail. In particular we wanted to understand how the ZULU ranking mechanism, which is based on predictions rates for randomly generated test suites, reflects the quality of Angluin-style implementations of automata learning.

Table 1. Algorithms: Configuration and Ranking

Algorithm	Dist. Set		Equivalence Query		Training	Rank
	Init.	Uniform	Continuous	Strategy	(Avg.)	
E.H.Blocking	$\{\varepsilon\}$	no	yes	block transitions	89.38	1
E.H.Weighted			yes	weight transitions	89.26	2
Random			no	random walks	88.93	6
run_random	$\{\varepsilon\} \cup \Sigma$	yes	no	random walks	80.17	14
run_blocking1			yes	block transitions	79.89	15
run_weighted1			yes	weight transitions	79.65	16

3.4 Discussion of the ZULU Rating Approach

In order to better understand the progress of the learning algorithms, let us consider some profile data of the training problem 49763507 in more detail, for which the ZULU environment allowed 8101 membership queries. Table 2 classifies

Table 2. Detailed Training Example: Problem 49763507

Algorithm	New Membership Queries			Rounds	States	Score
	Close Obs.	Analyze CEs	Search CEs			
E.H.Blocking	6,744	358	999	259	352	94.11
E.H.Weighted	6,717	349	1,035	262	351	94.61
Random	6,586	519	996	228	332	93.28
run_random	8,080	14	7	5	312	74.89
run_blocking1	8,074	11	16	6	319	73.06
run_weighted1	8,077	9	15	6	319	74.39

the consumption of these 8101 queries according to their usage during (1) the closure of the Observation Table, (2) the analysis of counterexamples, and (3) the search for counterexamples. Moreover, it shows the number of learning rounds performed, the detected states, and the eventual scores.

These results are quite drastic:

1. The difference between the two groups discussed in the previous section is even more impressive here. It is obvious that the first group profited from the extremely efficient search for counterexamples, which required in average only about 3 membership queries. In fact, the algorithms in the first group executed 50 times as many approximative equivalence queries as the algorithms of the second group.
2. The impact of the continuous equivalence queries, which make only a difference of 1,8% in the ZULU ranking (E.H.Blocking vs Random), make about 6% in the number of detected states, which is a lot, when considering the typical behavior of Angluin-style learning. Just consider the only 3% difference in the number of detected states between the Random options of the first group and the second group. In the ZULU rating they make a difference of 19%.
3. Despite the extremely different distribution of the allowed 8101 membership queries, the average number of membership queries required to obtain a new state seem quite similar. They range between 23 and 26. One should keep in mind, however, that the difficulty to find a new state strongly increases in the course of the algorithm. Thus having detected 8% more states is much.
4. The ZULU ranking seems to increase with the number of states. This interpretation is, however, wrong, as can be seen in Fig. 4 (for ZULU problem 85173129), where in the beginning, the quality of prediction drops from 60 to almost only 40 per cent! For the already discussed example (49763507) this effect is not as strong, but still the non-monotone developing of ratings throughout the learning process can be observed in Fig. 5. As we will discuss later, this effect, which shows here up already for randomly generated systems (in fact, we observed it in almost all Problems we treated as part of the ZULU challenge), gets worse when systems are man-made.

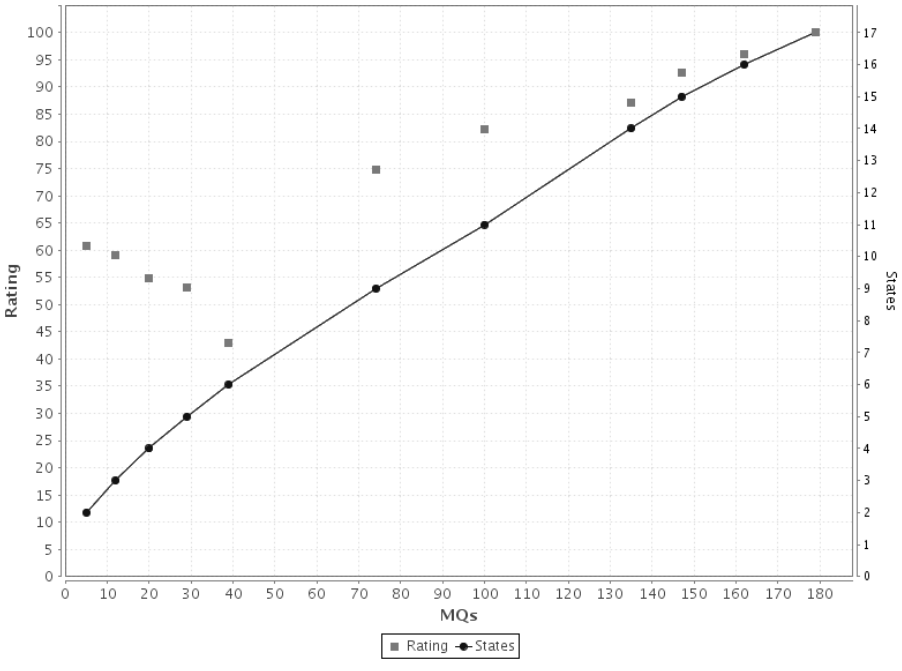


Fig. 4. ZULU Rating for all hypotheses in a learning attempt, Problem 85173129

5. There is a part of the learning process, where small improvements make a big difference in the ZULU ranking (e.g. for the two random versions, 13 more states made a difference of 19% in the ZULU score). This is the part after the chaotic behavior at the beginning and before the final phase, where progress gets exponentially harder.

The ZULU organizers also observed that the top teams had quite similar scores. Looking at Table 2 this is not too surprising, as much progress is needed at the end to achieve even marginal improvements for the ZULU rating (20 more in this stage very hard to detect states for 1,8% in the ZULU rating: see E.H.Blocking vs. Random of the first group). It was therefore decided to further reduce the number of allowed membership queries. This certainly diversified the results. However, it also increased the risk of algorithms being punished by the quality drop described under item 4 above.

We consider the ZULU competition as milestone for advancing the state of the art for practical learning. Still, we see a high potential for improvement. The following section describes RERS [8], a new initiative trying to push in this direction.

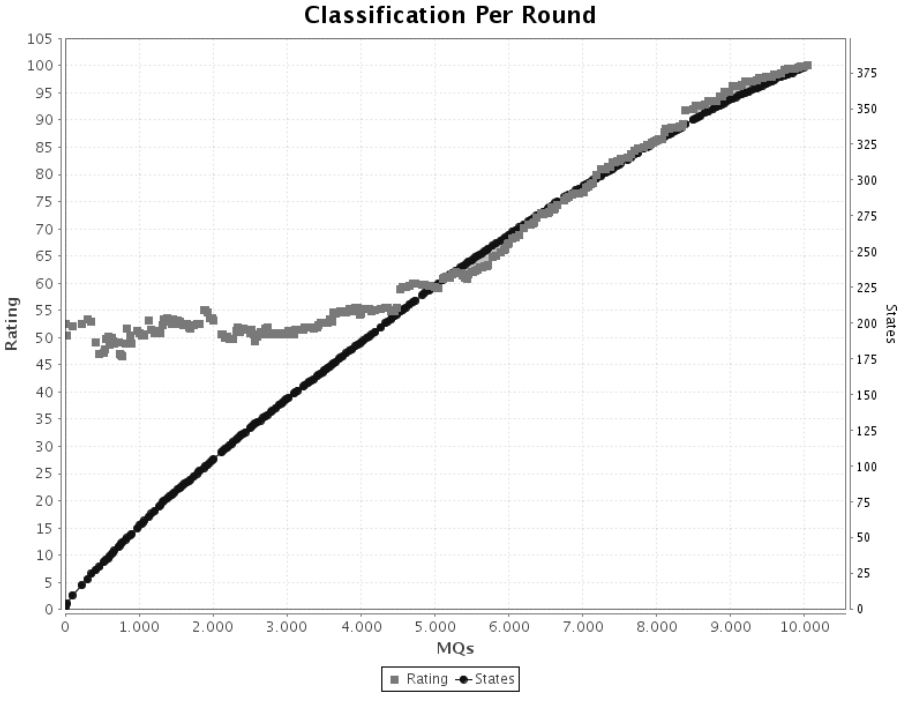


Fig. 5. ZULU Rating for all hypotheses in a learning attempt, Problem 49763507

4 RERS - Regular Inference of Reactive Systems

As the discussion of the results from the ZULU competition shows, an adequate quantification of a hypothesis' quality may depend on the domain of application: the test suite-based ZULU rating primarily addresses prediction quality concerning language membership. Perhaps surprisingly, this measure does not necessarily reflect the performance of Angluin-style implementations of learning algorithms. As we have seen, the prediction quality may quite significantly drop in the course of learning. The RERS initiative [8] aims at developing fair evaluation measurements tailored to specific application contexts. The discussion in the previous section already suggests discovered states as a good orientation for algorithmic progress. In fact, in our experience, discovered states are also a good measurement in the context of system testing [14,15]: The more system structure (in terms of states) is revealed, the better are the results of test generation algorithms.

Developing domain-specific notions of quality is, however, only one dimension of RERS, although a very important one, as it sets the standards for other dimensions, like:

- Establishing scenarios for realistic case studies,
- Developing methods making automata learning scalable,

– Providing technology to support the system interaction required for learning,
– Generalizing the notion of learning to capture new kinds of system behavior.

RERS does not only attempt to promote research in this direction, but intends to establish a framework in which progress in these dimensions can be measured, solutions can be compared, and experiences can be exchanged. Organizing challenges with branches capturing the currently most essential features is meant to be one of RERS prominent roles.

For next year we plan a challenge focusing on automata learning as a key technology for dealing with *black-box systems*, i.e., systems that can be observed, but for which no or little knowledge about the internal structure or even their intent is available. By this we intend to reveal the state of the art concerning (at least) the following kinds of scenarios, specifically for both, randomly generated systems (frequently used for evaluation, see e.g., the ZULU challenge), and man-made systems, the predominant case in practice:

1. Black-box systems with equivalence queries are not relevant in practice, but ideal for learning. Usually only approximations of equivalence queries can be generated.
2. For fast black-box systems (e.g. simulated ones) the number of membership queries is not as important as in other scenarios. While seemingly of only little practical use, some use cases, e.g., learning behavior of APIs, can come close.
3. In scenarios with black-box systems with high-cost membership queries (e.g., systems that are slow to respond or generate actual cost per query) it makes much sense to limit the amount of generated queries and to investigate ways how to decrease effective costs.
4. Scenarios where data values have to be considered are common and motivate the introduction of means to abstract from infinite value domains. While manually generated and fixed abstractions may be sufficient in many cases it may be necessary to refine such an abstraction during the learning process, if possible in an automated fashion according to system output.
5. Variants of non-deterministic or probabilistic systems. Such systems typically require more enhanced membership queries, which cannot be realized via normal system testing and therefore pose new implementation requirements (see e.g. [18]).

Central will be here the investigation of the practical limitations and the potential of *active* automata learning, which is characterized by its specific means of observation, i.e., its proactive way of posing membership queries and equivalence queries. ZULU has addressed the problem that equivalence queries are typically not realistic and need to be approximated via membership queries. However, also membership queries do not come for free but must be realized e.g. via testing in practice. The following subsections discuss important practical challenges according to the various characteristics of application scenarios, and illustrate that 'black does not equal black' in real-life black-box scenarios:

Interacting with real systems. The interaction with a realistic target system comes with two problems: A merely technical problem of establishing an adequate interface that allows one to apply test cases for realizing membership queries, and a conceptual problem of bridging the gap between the abstract learned model and the concrete runtime scenario.

The first problem is rather simple for systems designed for connectivity (e.g., Web-services or code libraries) which have a native concept of being invoked from the outside and come with documentation on how to accomplish this. Establishing connectivity may be arbitrarily complicated, however, for, e.g., some embedded systems which work within well-concealed environments and are only accessible via some proprietary GUI.

The second problem is conceptually more challenging. It concerns establishing an adequate abstraction level in terms of a communication alphabet, which on one hand leads to a useful model structure, but on the other hand also allows for an automatic back and forth translation between the abstract model level and the concrete target system level.

Membership queries. Whereas small learning experiments typically require only a few hundred membership queries, learning realistic systems may easily require several orders of magnitude more. This directly shows that the speed of the target system when processing membership queries, or as in most practical settings the corresponding test cases, is of the utmost importance. In contrast to simulation environments, which typically process several thousand of queries per second, real systems may well need many seconds or sometime even minutes per test case. In such a case, rather than parallelization, minimizing the number of required test cases is the key to success.

Reset. Active learning requires membership queries to be independent. Whereas this is no problem for simulated system, it may be quite problematic in practice. Solutions range here from reset mechanisms via homing sequences [26] or snapshots of the system state to the generation of independent fresh system scenarios. Indeed, in certain situations, executing each membership query with a separate independent user scenario may be the best one can do. Besides the overhead of establishing these scenarios, this also requires an adequate aggregation of the query results. E.g., the different user password combinations of the various used scenarios must be abstractly identified.

Parameters and value domains. Active learning classically is based on abstract communication alphabets. Parameters and interpreted values are only treated to an extend expressible within the abstract alphabet. In practice, this typically is not sufficient, not even for systems as simple as communication protocols, where, e.g., increasing sequence numbers must be handled, or where authentication requires matching user/password combinations. Due to the complexity of this problem, we do not expect any comprehensive solutions here. We rather think that domain- and problem-specific approaches must be developed in order to produce dedicated solutions.

Within RERS we want to provide the means for contestants to comparatively easily enter this exciting field of research, and to contribute to next year's RERS

challenge. Accordingly, the LearnLib (comprising libraries of algorithms, system connectors, algorithmic patterns, abstraction function etc.) will be made publicly available. This will allow contestants to start from an existing basis and extend it with new functionality.

5 Conclusion

We have summarized our experience with the ZULU challenge on active learning, presented our winning solution, investigated the character of ZULU's rating approach, and discussed how this approach can be taken further to establish a framework for the systematic investigation of domain-specific, scalable learning solutions for practically relevant application. In particular, we have discussed the RERS initiative, which provides a community platform together with a learning framework that allows users to interactively compose complex learning solutions on the basis of libraries for various learning components, system connectors, and other auxiliary functionality. This framework will be the backbone for an extended challenge on learning in 2011.

There are multiple dimensions to RERS, like the design of problems specific model structures, the development of flexible abstraction technologies, and the adequate treatment of parameters and values treatment. These need to be related to learning-specific technologies for e.g. treating counterexamples, realizing equivalence queries, or exploiting domain-specific knowledge about the target systems, and, a potential show stopper in practice, to adequate testing technology enabling the required kind of querying.

We therefore envisage to establish targeted sub-communities specialized and interested in specific scenarios. One such scenario could be the 'classical' learning scenario as addressed e.g. by the ZULU challenge, but there are many more, each of them with a particular learning profile. Examples range from the learning of I/O automata or Mealy machines for complex reactive systems or protocols, over the construction of (abstract) behavioral models of software components e.g. as a means to automate assume guarantee reasoning, to the detection of business processes by real life observation.

We hope that RERS will develop as a platform for synergetic cooperation, that the organized challenges will help to identify the strength and weaknesses of the various solutions and their mutual interplay, and that the possibility for easy experimentation, e.g. using the LearnLib, will make automata learning a convenient tool in many application contexts.

References

1. Aarts, F., Schmaltz, J., Vaandrager, F.: Inference and abstraction of the biometric passport. In: ISoLA 2010, Part I. LNCS, vol. 6415, pp. 681–694. Springer, Heidelberg (2010)
2. Angluin, D.: Learning Regular Sets from Queries and Counterexamples. Information and Computation 75(2), 87–106 (1987)

3. Balcázar, J.L., Díaz, J., Gavaldà, R.: Algorithms for Learning Finite Automata from Queries: A Unified View. In: Advances in Algorithms, Languages, and Complexity, pp. 53–72 (1997)
4. Berg, T., Grinchtein, O., Jonsson, B., Leucker, M., Raffelt, H., Steffen, B.: On the Correspondence Between Conformance Testing and Regular Inference. In: Cerioli, M. (ed.) FASE 2005. LNCS, vol. 3442, pp. 175–189. Springer, Heidelberg (2005)
5. Bohlin, T., Jonsson, B., Soleimanifard, S.: Inferring compact models of communication protocol entities. In: ISoLA 2010, Part I. LNCS, vol. 6415, pp. 666–680. Springer, Heidelberg (2010)
6. Bollig, B., Katoen, J.-P., Kern, C., Leucker, M.: Smyle: A Tool for Synthesizing Distributed Models from Scenarios by Learning. In: van Breugel, F., Chechik, M. (eds.) CONCUR 2008. LNCS, vol. 5201, pp. 162–166. Springer, Heidelberg (2008)
7. Broy, M., Jonsson, B., Katoen, J.-P., Leucker, M., Pretschner, A.: Model-Based Testing of Reactive Systems. LNCS, vol. 3472. Springer, Heidelberg (2005)
8. TU Dortmund Chair of Programming Systems, Department of Computer Science. RERS - A Challenge In Active Learning, `http://leo.cs.tu-dortmund.de:8100/` (version from 20.06.2010)
9. Chow, T.S.: Testing Software Design Modeled by Finite-State Machines. IEEE Trans. on Software Engineering 4(3), 178–187 (1978)
10. Cobleigh, J.M., Giannakopoulou, D., Păsăreanu, C.S.: Learning assumptions for compositional verification. In: Garavel, H., Hatcliff, J. (eds.) TACAS 2003. LNCS, vol. 2619, pp. 331–346. Springer, Heidelberg (2003)
11. Combe, D., de la Higuera, C., Zulu, Jean-Christophe, J.: an Interactive Learning Competition. In: Proceedings of FSMNLP 2009 (to appear, 2010)
12. Combe, D., de la Higuera, C., Janodet, J.-C., Ponge, M.: Zulu - Active learning from queries competition, `http://labh-curien.univ-st-etienne.fr/zulu/index.php` (version from 01.08.2010)
13. Fujiwara, S., von Bochmann, G., Khendek, F., Amalou, M., Ghedamsi, A.: Test Selection Based on Finite State Models. IEEE Trans. on Software Engineering 17(6), 591–603 (1991)
14. Hagerer, A., Hungar, H., Niese, O., Steffen, B.: Model generation by moderated regular extrapolation. In: Kutsche, R.-D., Weber, H. (eds.) FASE 2002. LNCS, vol. 2306, pp. 80–95. Springer, Heidelberg (2002)
15. Hagerer, A., Margaria, T., Niese, O., Steffen, B., Brune, G., Ide, H.-D.: Efficient regression testing of CTI-systems: Testing a complex call-center solution. Annual review of communication, Int. Engineering Consortium (IEC) 55, 1033–1040 (2001)
16. Hungar, H., Niese, O., Steffen, B.: Domain-specific optimization in automata learning. In: Hunt Jr., W.A., Somenzi, F. (eds.) CAV 2003. LNCS, vol. 2725, pp. 315–327. Springer, Heidelberg (2003)
17. Kearns, M.J., Vazirani, U.V.: An Introduction to Computational Learning Theory. MIT Press, Cambridge (1994)
18. Kwiatkowska, M.Z., Norman, G., Parker, D., Qu, H.: Assume-Guarantee Verification for Probabilistic Systems. In: Esparza, J., Majumdar, R. (eds.) TACAS 2010. LNCS, vol. 6015, pp. 23–37. Springer, Heidelberg (2010)
19. Maler, O., Pnueli, A.: On the Learnability of Infinitary Regular Sets. Information and Computation 118(2), 316–326 (1995)
20. Margaria, T., Niese, O., Raffelt, H., Steffen, B.: Efficient test-based model generation for legacy reactive systems. In: HLDVT 2004: Proceedings of the Ninth IEEE International on High-Level Design Validation and Test Workshop, Washington, DC, USA, pp. 95–100. IEEE Computer Society, Los Alamitos (2004)

21. Nerode, A.: Linear Automaton Transformations. Proceedings of the American Mathematical Society 9(4), 541–544 (1958)
22. Pasareanu, C.S., Giannakopoulou, D., Bobaru, M.G., Cobleigh, J.M., Barringer, H.: Learning to divide and conquer: applying the L* algorithm to automate assume-guarantee reasoning. Formal Methods in System Design 32(3), 175–205 (2008)
23. Peled, D., Vardi, M.Y., Yannakakis, M.: Black Box Checking. In: Wu, J., Chanson, S.T., Gao, Q. (eds.) Proc. FORTE 1999, pp. 225–240. Kluwer Academic, Dordrecht (1999)
24. Raffelt, H., Merten, M., Steffen, B., Margaria, T.: Dynamic testing via automata learning. Int. J. Softw. Tools Technol. Transf. 11(4), 307–324 (2009)
25. Raffelt, H., Steffen, B., Berg, T., Margaria, T.: LearnLib: a framework for extrapolating behavioral models. Int. J. Softw. Tools Technol. Transf. 11(5), 393–407 (2009)
26. Rivest, R.L., Schapire, R.E.: Inference of finite automata using homing sequences. Inf. Comput. 103(2), 299–347 (1993)
27. Shahbaz, M., Groz, R.: Inferring Mealy Machines. In: Cavalcanti, A., Dams, D.R. (eds.) FM 2009. LNCS, vol. 5850, pp. 207–222. Springer, Heidelberg (2009)
28. Shahbaz, M., Li, K., Groz, R.: Learning Parameterized State Machine Model for Integration Testing. In: Proc. 31th Annual Int. Computer Software and Applications Conf., Washington, DC, USA, vol. 2, pp. 755–760. IEEE Computer Society, Los Alamitos (2007)

Author Index