Tiziana Margaria Bernhard Steffen (Eds.)

LNCS 8803

Leveraging Applications of Formal Methods, Verification and Validation

Specialized Techniques and Applications

6th International Symposium, ISoLA 2014
Imperial, Corfu, Greece, October 8–11, 2014
Proceedings, Part II

2 Part II

Springer

Tiziana Margaria Bernhard Steffen (Eds.)

Leveraging Applications of Formal Methods, Verification and Validation

Specialized Techniques and Applications

6th International Symposium, ISoLA 2014
Imperial, Corfu, Greece, October 8-11, 2014
Proceedings, Part II

Springer

Volume Editors

Tiziana Margaria
University of Limerick, Ireland
E-mail: tiziana.margaria@lero.ie

Bernhard Steffen
TU Dortmund, Germany
E-mail: steffen@cs.tu-dortmund.de

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-662-45230-1 e-ISBN 978-3-662-45231-8
DOI 10.1007/978-3-662-45231-8
Springer Heidelberg New York Dordrecht London

Library of Congress Control Number: Applied for

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

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Introduction

Welcome to the proceedings of ISoLA 2014, the 6th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation, that was held in Imperial, Corfu (Greece) during October 8–11, 2014, endorsed by EASST, the European Association of Software Science and Technology.

This year's event was at the same time ISoLA's tenth anniversary. It also followed the tradition of its symposia forerunners held 2004 and 2006 in Cyprus, 2008 in Chalkidiki, and 2010 as well as 2012 in Crete, and the series of ISoLA Workshops in Greenbelt (USA) in 2005, Poitiers (France) in 2007, Potsdam (Germany) in 2009, in Vienna (Austria) in 2011, and 2013 in Palo Alto (USA).

As in the previous editions, ISoLA 2014 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, test, and maintenance of systems from the point of view of their different application domains. Thus, since 2004 the ISoLA series of events serves the purpose of bridging the gap between designers and developers of rigorous tools on one side, and users in engineering and in other disciplines on the other side. It fosters and exploits synergetic relationships among scientists, engineers, software developers, decision makers, and other critical thinkers in companies and organizations. By providing a specific, dialogue-oriented venue for the discussion of common problems, requirements, algorithms, methodologies, and practices, ISoLA aims in particular at supporting researchers in their quest to improve the usefulness, reliability, flexibility, and efficiency of tools for building systems, and users in their search for adequate solutions to their problems.

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

- Statistical Model Checking, Past Present and Future (K. Larsen, A. Legay)
- Formal Methods and Analysis in Software Product Line Engineering (I. Schäfer, M. ter Beck)
- Risk-Based Testing (M. Felderer, M. Wendland, I. Schieferdecker)
- Scientific Workflows (J. Kok, A. Lamprecht, K. Turner, K. Wolstencroft)
- Medical Cyber Physical Systems (E. Bartocci, S. Gao, S. Smolka)
- Evaluation and Reproducibility of Program Analysis (M. Schordan, W. Lowe, D. Beyer)
- Automata Learning (F. Howar, B. Steffen)
- Rigorous Engineering of Autonomic Ensembles (R. de Nicola, M. Hölzl, M. Wirsing)
- Engineering Virtualized Services (R. Hähnle, E. Broch Johnsen)
- Security and Dependability for Resource Constrained Embedded Systems (B. Hamid, C. Rudolph)
- Semantic Heterogeneity in the Formal Development of Complex Systems (I. Ait Sadoune, J.P. Gibson)

- Evolving Critical Systems (M. Hinchey, T. Margaria)
- Model-Based Code-Generators and Compilers (J. Knoop, W. Zimmermann, U. Assmann)
- Processes and Data Integration in the Networked Healthcare (J. Mündler, T. Margaria, C. Rasche)

The symposium also featured:

- Tutorial: Automata Learning in Practice (B. Steffen, F. Howar)
- RERS: Challenge on Rigorous Examination of Reactive Systems (F. Howar, J. van de Pol, M. Schordan, M. Isberner, T. Ruys, B. Steffen)
- Doctoral Symposium and Poster Session (A.-L. Lamprecht)
- Industrial Day (A. Hessenkämper)

Co-located with the ISoLA Symposium was:

- STRESS 2014 - Third International School on Tool-Based Rigorous Engineering of Software Systems (J. Hatcliff, T. Margaria, Robby, B. Steffen)

We thank the track organizers, the members of the Program Committee and their subreferees for their effort in selecting the papers to be presented, the local organization chair, Petros Stratis, and the Easyconference team for their continuous precious support during the week as well as during the entire two-year period preceding the events. We also thank Springer for being, as usual, a very reliable partner for the proceedings production. Finally, we are grateful to Horst Voigt for his Web support, and to Dennis Kühn, Maik Merten, Johannes Neubauer, and Stephan Windmüller for their help with the online conference service (OCS).

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

October 2014 Tiziana Margaria
Bernhard Steffen

Organization

Symposium Chair

Bernhard Steffen

Program Chair

Tiziana Margaria

Program Committee:

Yamine Ait Ameur	ISAE-ENSMA, France
Idi Ait-Sadoune	SUPÉLEC, France
Uwe Assmann	TU Dresden, Germany
Ezio Bartocci	TU Wien, Austria
Dirk Beyer	University of Passau, Germany
Rocco De Nicola	IMT Lucca, Italy
Michael Felderer	University of Innsbruck, Austria
Sicun Gao	Carnegie Mellon University, USA
J. Paul Gibson	Télécom SudParis, France
Kim Guldstrand Larsen	Aalborg University, Denmark
Reiner Hähnle	TU Darmstadt, Germany
Brahim Hamid	IRIT, France
Mike Hinchey	Lero, Ireland
Matthias Hölzl	Ludwig-Maximilians-University Munich, Germany
Falk Howar	Carnegie Mellon University, USA
Einar Broch Johnsen	University of Oslo, Norway
Jens Knoop	TU Wien, Austria
Joost Kok	LIACS Leiden University, The Netherlands
Anna-Lena Lamprecht	University of Potsdam, Germany
Axel Legay	Inria, France
Welf Löwe	Linnaeus University, Sweden
Tiziana Margaria	University of Limerick, Ireland
Christoph Rasche	University of Potsdam, Germany
Carsten Rudolph	Fraunhofer SIT, Germany
Ina Schäfer	TU Braunschweig, Germany
Ina Schieferdecker	FU Berlin, Germany
Markus Schordan	Lawrence Livermore National Laboratory, USA

Scott Smolka	State University of New York at Stony Brook, USA
Bernhard Steffen	TU Dortmund, Germany
Maurice ter Beek	Institute of the National Research Council of Italy CNR, Italy
Kenneth Turner	University of Stirling, UK
Marc-Florian Wendland	Fraunhofer Institut, Germany
Martin Wirsing	Ludwig-Maximilians-University Munich, Germany
Katy Wolstencroft	LIACS Leiden University, The Netherlands
Wolf Zimmermann	Martin-Luther-University Halle-Wittenberg, Germany

Table of Contents – Part II

Table of Contents – Part I

LNCS Transactions on Foundations for Mastering Change

Introduction to Track on Engineering Virtualized Services[⋆]

Reiner Hähnle[1] and Einar Broch Johnsen[2]

[1] Technical University of Darmstadt, Germany
haehnle@cs.tu-darmstadt.de
[2] Dept. of Informatics, University of Oslo, Norway
einarj@ifi.uio.no

Abstract. Virtualization is a key technology enabler for cloud computing. Despite the added value and compelling business drivers of cloud computing, this new paradigm poses considerable new challenges that have to be addressed to render its usage effective for industry. Virtualization makes elastic amounts of resources available to application-level services; for example, the processing capacity allocated to a service may be changed according to demand. Current software development methods, however, do not support the modeling and validation of services running on virtualized resources in a satisfactory way. This seriously limits the potential for fine-tuning services to the available virtualized resources as well as for designing services for scalability and dynamic resource management. The track on *Engineering Virtualized Services* aims to discuss key challenges that need to be addressed to enable software development methods to target resource-aware virtualized services.

1 Moving into the Clouds

The planet's data storage and processing is about to move into the clouds. This has the potential to revolutionize how we will interact with computers in the future. Although the privacy of data stored in the cloud remains a challenge, cloud-based data processing, or cloud computing, is already emerging as an economically interesting business model, due to an undeniable added value and compelling business drivers [5]. One such driver is *elasticity*: businesses pay for computing resources when they are needed, instead of provisioning in advance with huge upfront investments. New resources such as processing power or memory can be added to the cloud's virtual computers on the fly, or additional virtual computers can be provided to the client application. Going beyond shared storage, the main potential in cloud computing lies in its scalable virtualized framework for data processing. If a service uses cloud-based processing, its capacity can be automatically adjusted when new users arrive. Another driver is *agility*: new services can be deployed quickly and flexibly on the market at limited cost. This allows a service to handle its end-users in a flexible manner without requiring initial investments in hardware before the service can be launched.

[⋆] Partly funded by the EU project FP7-610582 ENVISAGE: Engineering Virtualized Services (`http://www.envisage-project.eu`).

Reliability and control of resources are barriers to the industrial adoption of cloud computing today. To overcome these barriers and to gain control of the virtualized resources on the cloud, client services need to become resource-aware. Looking beyond today's cloud, we may then expect *virtualized services* which dynamically combine distributed and heterogeneous resources from providers of utility computing in an increasingly fine-grained way. Making full usage of the potential of virtualized computation requires that we rethink the way in which we design and develop software.

2 Empowering the Designer

The elasticity of software executed in the cloud means that its designers are given far reaching control over the resource parameters of the execution environment, such as the number and kind of processors, the amount of memory and storage capacity, and the bandwidth. In principle, these parameters can even be changed dynamically, at runtime. This means that the client of a cloud service not only can deploy and run software, but is also in full control of the trade-offs between the incurred cost and the delivered quality-of-service.

To exploit these new possibilities, software in the cloud must be *designed for scalability*. Today, software is often designed based on specific assumptions about deployment, such as the size of data structures, the amount of random access memory, the number of processors. Rescaling usually requires extensive design changes when scalability has not been taken into account from the start. This consideration makes it clear that it is essential to detect and fix deployment errors, such as the impossibility to meet a service level agreement, already *in the design phase*. To make full usage of the opportunities of cloud computing, software development for the cloud demands a design methodology that

– can take into account deployment modeling at early design stages and
– permits the detection of deployment errors early and efficiently, preferably using software tools, such as simulators, test generators, and static analyzers.

Clearly, there is a new *software engineering challenge* which needs to be addressed: how can the validation of deployment decisions be pushed up to the modeling phase of the software development chain without convoluting the design with deployment details?

3 Controlling Deployment in the Design Phase

When a service is developed today, the developers first design its *functionality*, then they determine which resources are needed for the service, and ultimately the *provisioning* of these resources is controlled through a *service level agreement* (SLA). So far, these three parts of a deployed cloud service tend to live in separate worlds. It is important to bridge the gaps between them.

The first gap is between the client layer functionality and the provisioning layer. It can be closed by a virtualization interface which allows the client layer to read and change resource parameters. The second gap is between SLAs and the client layer. Here the key observation is that the service contract part of an SLA can be formalized as a specification contract with rigorous semantics. This enables formal analysis of the client behavior with respect to the SLA *at design time*. Possible analyses include resource consumption, performance analysis, test case generation, and formal verification [2]. For suitable modeling and specification languages such analyses can be highly automated [3].

4 The Papers

The ISoLA track *Engineering Virtualized Services*, organized in the context of the EU FP7 project ENVISAGE, reflects the aims laid down above and focuses on systematic and formal approaches to

- modeling services deployed on the cloud,
- formalizing SLAs, and
- analysis of SLAs.

A crucial aspect in modeling cloud services is the handling of faults and errors. This is addressed in the papers by Göri et al. [9] and Lanese et al. [10] The former critically discusses the different choices that have to be made when defining a fault model for concurrent, actor-based languages. The latter proposes a specific failure model for concurrent objects with cooperative scheduling that automatically re-establishes object invariants after program failures, thereby eliminating the need to manually write this problematic code. The paper by De Boer & Nobakht [7] shows that high-level modeling of services can be achieved already on the JAVA level by embedding an actor-based API with the help of lambda expressions and extended dynamic invocation support, which is available since JAVA 8.

There are two papers on formalization of SLAs and service contracts: the paper by Woodcock et al. [11] describes the COMPASS Modelling Language CML, which is used to formally model large-scale Systems of Systems and the contracts which bind them together. The paper by Causevic et al. [6] presents the REMES HDCL language by way of a case study that formalizes service negotiation in a distributed energy management scenario.

Moving to analysis, a central aspect in deployment of cloud services is to obtain reliable estimates on resource consumption and, hence, on adequate provisioning. The paper by Giachino & Laneve [8] suggests a type system for a concurrent, object-oriented language that permits dynamic scaling out and scaling in. The type of a program is behavioural and it reflects the resource deployments over periods of time. On the other hand, the paper by Albert et al. [1] focuses on automatic inference of upper bounds for the amount of data transmissions that may occur in a distributed system. Finally, the paper by Bubel et al. [4] concentrates on formal verification of functional aspects of service contracts: it presents a technique for compositional verification in presence of constant evolutionary changes to the verification target.

Taken together, the eight papers in this track comprise an exciting snapshot of the state-of-art in formal approaches to modeling services deployed on the cloud as well as to formalization and analysis of SLAs.

References

1. Albert, E., Correas, J., Martin-Martin, E., Román-Díez, G.: Static inference of transmission data sizes in distributed systems. In: Margaria, T., Steffen, B. (eds.) ISoLA 2014, Part II. LNCS, vol. 8803, pp. 104–119. Springer, Heidelberg (2014)
2. Albert, E., de Boer, F., Hähnle, R., Johnsen, E.B., Laneve, C.: Engineering virtualized services. In: Babar, M.A., Dumas, M. (eds.) 2nd Nordic Symposium on Cloud Computing & Internet Technologies (NordiCloud 2013), pp. 59–63. ACM (2013)
3. Albert, E., de Boer, F., Hähnle, R., Johnsen, E.B., Schlatte, R., Tapia Tarifa, S.L., Wong, P.Y.H.: Formal modeling of resource management for cloud architectures: An industrial case study. Journal of Service-Oriented Computing and Applications (2013) (Springer Online First), doi:10.1007/s11761-013-0148-0
4. Bubel, R., Hähnle, R., Pelevina, M.: Fully abstract operation contracts. In: Margaria, T., Steffen, B. (eds.) ISoLA 2014, Part II. LNCS, vol. 8803, pp. 120–134. Springer, Heidelberg (2014)
5. Buyya, R., Yeo, C.S., Venugopal, S., Broberg, J., Brandic, I.: Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. Future Generation Computer Systems 25(6), 599–616 (2009)
6. Čaušević, A., Seceleanu, C., Pettersson, P.: Distributed energy management case study: A formal approach to analyzing utility functions. In: Margaria, T., Steffen, B. (eds.) ISoLA 2014, Part II. LNCS, vol. 8803, pp. 74–87. Springer, Heidelberg (2014)
7. Nobakht, B., de Boer, F.S.: Programming with actors in Java 8. In: Margaria, T., Steffen, B. (eds.) ISoLA 2014, Part II. LNCS, vol. 8803, pp. 37–53. Springer, Heidelberg (2014)
8. Giachino, E., Laneve, C.: Towards the typing of resource deployment. In: Margaria, T., Steffen, B. (eds.) ISoLA 2014, Part II. LNCS, vol. 8803, pp. 88–103. Springer, Heidelberg (2014)
9. Göri, G., Johnsen, E.B., Schlatte, R., Stolz, V.: Erlang-style error recovery for concurrent objects with cooperative scheduling. In: Margaria, T., Steffen, B. (eds.) ISoLA 2014, Part II. LNCS, vol. 8803, pp. 5–21. Springer, Heidelberg (2014)
10. Lanese, I., Lienhardt, M., Bravetti, M., Johnsen, E.B., Schlatte, R., Stolz, V., Zavattaro, G.: Fault model design space for cooperative concurrency. In: Margaria, T., Steffen, B. (eds.) ISoLA 2014, Part II. LNCS, vol. 8803, pp. 22–36. Springer, Heidelberg (2014)
11. Woodcock, J., Cavalcanti, A., Fitzgerald, J., Foster, S., Larsen, P.G.: Contracts in CML. In: Margaria, T., Steffen, B. (eds.) ISoLA 2014, Part II. LNCS, vol. 8803, pp. 54–73. Springer, Heidelberg (2014)

Erlang-Style Error Recovery for Concurrent Objects with Cooperative Scheduling[⋆]

Georg Göri[1], Einar Broch Johnsen[2], Rudolf Schlatte[2], and Volker Stolz[2]

[1] University of Technology, Graz, Austria
goeri@student.tugraz.at
[2] University of Oslo, Norway
{einarj,rudi,stolz}@ifi.uio.no

Abstract. Re-establishing a safe program state after an error occurred is a known problem. Manually written error-recovery code is both more difficult to test and less often executed than the main code paths, hence errors are prevalent in these parts of a program. This paper proposes a failure model for concurrent objects with cooperative scheduling that automatically re-establishes object invariants after program failures, thereby eliminating the need to manually write this problematic code. The proposed model relies on a number of features of actor-based object-oriented languages, such as asynchronous method calls, co-operative scheduling with explicit synchronization points, and communication via future variables. We show that this approach can be used to implement Erlang-style process linking, and implement a supervision tree as a proof-of-concept.

1 Introduction

Crashes and errors in real-world systems are not always due to faulty programming. Especially but not only in distributed systems, error conditions can arise that are not a consequence of the logic of the running program. Robust systems must be able to deal with and mitigate such unexpected conditions. At the same time, error recovery code is notoriously hard to test.

An influential approach to more robust systems is "Crash-Only Software" [4], i.e., letting system components fail and restarting them. Erlang [2,19] is a widely-used functional language which successfully adopts these ideas. However, inherent in such subsystem restarts is the accompanying loss of state. This is much less a problem with programs written in a functional style than with programs written using object-oriented techniques, where the objects themselves hold state. This paper describes an approach to crash-only software which can keep objects alive without explicit code to restore object invariants.

The approach of this paper is based on concurrent objects which communicate by means of *asynchronous method calls*; the caller allocates a *future* as a container for the forthcoming result of the method call, and keeps executing until the result of the call is needed. Since execution can get stuck waiting for a reply,

⋆ Partially funded by the EU project FP7-610582 ENVISAGE: Engineering Virtualized Services (`http://www.envisage-project.eu`).

we allow process execution to suspend by introducing processor release points related to the polling of futures. Scheduling is *cooperative* via release-points in the code, awaiting either a condition on the state of the object, or the availability of the result from a method call. To concretize the approach, we use some features from the abstract behavior specification language ABS [11], a statically typed object-oriented modeling language targeting distributed systems. ABS has a formal semantics implemented in the rewriting logic tool Maude [7], which can be used to explore the runtime behavior of specifications.

This paper introduces linguistic means to both abort a single computation without corrupting object state and to terminate an object with all its pending processes. We provide a formal semantics for how those faults propagate through asynchronous communication. Callers may decide to not care about faults and fail themselves when trying to access the result of a call whose computation aborted, or use a safe means of access that allows them to explicitly distinguish a fault from a normal result and react accordingly. We show the usefulness of the new language primitives by showing how they allow us to implement process linking and supervision hierarchies, the standard recovery features of Erlang.

The rest of the paper is organized as follows. Section 2 describes the ABS language, Section 3 the novel failure model. Section 4 presents an operational semantics of a subset of the language, and illustrates the new functionality by modeling Erlang's well-known supervision architecture, and Section 5 discusses related and future work.

2 Behavioral Modeling in ABS

ABS is an abstract, executable, object-oriented modeling language with a formal semantics [11], targeting distributed systems. ABS is based on concurrent objects [5, 13] communicating by means of asynchronous method calls. Objects in ABS support interleaved concurrency based on explicit scheduling points. This allows active and reactive behavior to be easily combined, by means of a cooperative scheduling of processes which stem from method calls. Asynchronous method calls and cooperative scheduling allow the verification of distributed and concurrent programs by means of sequential reasoning [8]. In ABS this is reflected in a proof system for local reasoning about objects where the class invariant must hold at all scheduling points [9].

ABS combines functional and imperative programming styles with a Java-like syntax. Objects execute in parallel and communicate through asynchronous method calls. However, the data manipulation inside methods is modeled using a simple functional language based on user-defined algebraic data types and functions. Thus, the modeler may abstract from the details of low-level imperative implementations of data structures while maintaining an overall object-oriented design close to the target system.

The Functional Layer. The functional layer of ABS consists of algebraic data types such as the empty type `Unit`, booleans `Bool`, and integers `Int`; parametric

$$\begin{aligned}
T &::= I \mid D \mid D\langle\overline{T}\rangle\\
A &::= X \mid T \mid D\langle\overline{A}\rangle\\
Dd &::= \textbf{data}\ D[\langle\overline{A}\rangle] = [\overline{Cons}];\\
Cons &::= Co[(\overline{A})]\\
F &::= \textbf{def}\ A\ fn[\langle\overline{A}\rangle](\overline{A\ x}) = e;\\
e &::= x \mid v \mid Co[(\overline{e})] \mid fn(\overline{e}) \mid \textbf{case}\ e\ \{\overline{br}\}\\
v &::= Co[(\overline{v})] \mid \textbf{null}\\
br &::= p \Rightarrow e;\\
p &::= \_ \mid x \mid v \mid Co[(\overline{p})]
\end{aligned}$$

$$\begin{aligned}
P &::= \overline{IF}\ \overline{CL}\ \{[\overline{T\ x};]\ s\}\\
IF &::= \textbf{interface}\ I\ \{\,[\overline{Sg}]\,\}\\
CL &::= \textbf{class}\ C\ [(\overline{T\ x})]\ [\textbf{implements}\ \overline{I}]\ \{\,[\overline{T\ x};]\ \overline{M}\}\\
Sg &::= T\ m\ ([\overline{T\ x}])\\
M &::= Sg\ \{[\overline{T\ x};]\ s\,\}\\
g &::= b \mid x? \mid g \wedge g\\
s &::= s;s \mid \textbf{skip} \mid \textbf{if}\ b\ \{\,s\,\}\ [\textbf{else}\ \{\,s\,\}] \mid \textbf{while}\ b\ \{\,s\,\}\\
&\quad \mid \textbf{suspend} \mid \textbf{await}\ g \mid x = rhs \mid \textbf{return}\ e\\
rhs &::= e \mid cm \mid \textbf{new}\ C\ (\overline{e})\\
cm &::= [e]!m(\overline{e}) \mid x.\textbf{get}
\end{aligned}$$

Fig. 1. ABS syntax for the functional (left) and imperative (right) layers. The terms $\overline{e}$ and $\overline{x}$ denote possibly empty lists over the corresponding syntactic categories, and square brackets [ ] optional elements.

data types such as sets `Set<X>` and maps `Map<X>` (for a type parameter `X`); and functions over values of these data types, with support for pattern matching.

The syntax of the functional layer is given in Figure 1 (left). The ground types T are interfaces I, type names D, and instantiated parametric data types $D\langle\overline{T}\rangle$. Parametric data types A allow type names to be parameterized by type variables X. User-defined data types definitions Dd introduce a name D for a new data type, parameters $\overline{A}$, and a list of constructors $Cons$. User-defined function definitions F have a return type A, a name fn, possible type parameters, a list of typed input variables x, and an expression e. Expressions e are variables x, values v, constructor, functional, and case expressions. Values v are constructors applied to values, or `null`. Case expressions match an expression e to a list of case branches br on the form $p \Rightarrow e$ which associate a pattern p with an expression e. Branches are evaluated in the listed order, the (possibly nested) pattern p includes an underscore which works as a wild card during pattern matching; variables in `p` are bound during pattern matching and are in the scope of the branch expression e. ABS provides a library with standard data types such as booleans, integers, sets, and maps, and functions over these data types.

The functional layer of ABS can be illustrated by considering naive *polymorphic sets* defined using a type variable `X` and two constructors `EmptySet` and `Insert`:

```
data Set<X> = EmptySet | Insert(X, Set<X>);
```

Two functions `contains`, which checks whether an item `el` is an element in a set `set`, and `take`, which selects an element from a non-empty set `set`, can be defined by pattern matching over `set`:

```
def Bool contains<X>(Set<X> set, X el) =
  case set {
    EmptySet => False ;
    Insert(el, _) => True;
    Insert(_, xs) => contains(xs, el); };

def X take<X>(Set<X> set) = case set { Insert(e, _) => e; };
```

The Imperative Layer. The imperative layer of ABS addresses concurrency, communication, and synchronization at the level of objects, and defines interfaces, classes, and methods. In contrast to mainstream object-oriented languages, ABS does not have an explicit concept of threads. Instead a thread of execution is unified with an object as the unit of concurrency and distribution, which eliminates race conditions in the models. Objects are *active* in the sense that their `run` method, if defined, gets called upon creation.

The syntax of the imperative layer of ABS is given in Figure 1 (right). A program P lists interface definitions IF and class definitions CL, and has a main block $\{[\overline{T}\ \overline{x};]\ s\}$ where the variables x of types T are in the scope of the statement s. Interface and class definitions, as well as signatures Sg and method definitions M are as in Java. As usual, **this** is a read-only field of an object, referring to the identifier of the object; similarly, we let **destiny** be a read-only variable in the scope of a method activation, referring to the future for the return value from the method activation. Below we focus on explaining the asynchronous communication and suspension mechanisms of ABS.

Communication and synchronization are decoupled in ABS. Communication is based on asynchronous method calls, denoted by assignments `f=o!m(e)` where `f` is a future variable, `o` an object expression, and `e` are (data value or object) expressions. After calling `f=o!m(e)`, the caller may proceed with its execution *without blocking* on the method reply. Two operations on future variables control synchronization in ABS. First, the statement **await** `f?` *suspends the active process* unless a return value from the call associated with `f` has arrived, allowing other processes in the same object to execute. Second, the return value is retrieved by the expression `f.`**get**, which *blocks all execution in the object* until the return value is available. Inside an object, ABS also supports standard synchronous method calls `o.m(e)`.

Objects locally sequentialize execution, resembling a monitor with release points but without explicit signaling. An object can have at most one active process. This active process can be unconditionally suspended by the statement **suspend**, adding this process to the queue of the object, from which an enabled process is then selected for execution. The guards `g` in **await** `g` control suspension of the active process and consist of Boolean conditions b conjoined with return tests `f?` on future variables `f` and with time-bounded suspensions `duration(e1,e2)` which become enabled between a best-case `e1` and a worst-case `e2` amount of time. Just like functional expressions, guards `g` are side-effect free. Instead of suspending, the active process may *block* while waiting for a reply as discussed above, or it may block for some amount of time between a best-case `e1` and a worst-case `e2`, using the syntax `duration(e1,e2)` [3]. The remaining statements of ABS are standard; e.g., sequential composition $s_1; s_2$, assignment `x=rhs`, and **skip**, **if**, **while**, and **return** constructs. Right hand side expressions `rhs` include the creation of an object **new** `C(e)`, method calls, and future dereferencing `f.`**get**, in addition to the functional expressions `e`.

Example. To illustrate the imperative layers of ABS, let us consider an interface `Account`, with methods `deposit` and `withdraw`, which is implemented by a class

```
interface Account {
    Unit deposit (Int amount);
    Unit withdraw (Int amount);
}

class Account implements Account {
    List<Int> transactions = Nil; // log of transactions
    Int balance = 0; // current balance
    Unit deposit (Int amount) {
        transactions = Cons(amount, transactions);
        balance = balance + amount;
    }
    Unit withdraw (Int amount) {
        transactions = Cons(-amount, transactions);
        if (balance < amount) abort "Insufficient funds";
        balance = balance - amount;
    }
}
```

Fig. 2. Bank account with history in ABS

`BankAccount` (as shown in Figure 2). We see that expressions from the functional layer are used inside the method implementations; e.g., the constructor `Cons` is used in the right hand side of an assignment to extend the list of transactions, and infix functions `+` and `-` are similarly used to adjust the balance.

To approach the theme of the next section, the example does not resolve the case of negative balance on the account (ignoring the issue of a better design which checks the condition before updating the history). A call to `withdraw` will only succeed if the balance is sufficient; if the `balance` is less than `amount` it is unclear what would be meaningful behavior in order to restore a class invariant like `balance`$\geq$`0`, and the method activation will *abort*: the previous state of the object will be restored, and the future storing the implicit return value of `Unit` type will be filled with a value indicating that an error occurred.

3 Failure Models and Error handling

Apart from user-specified aborts, it is very common for programs to run into so-called *runtime errors*, i.e., abnormal termination in a case where the operational semantics does not prescribe how the system can proceed. Prominent representatives of this class of faults are division by zero, null pointer accesses in languages that allow pointer dereferences, and errors that are propagated from the runtime system in managed languages, like out of memory errors when no more objects can be allocated.

In the semantics of the ABS language, behavior in those situations is underspecified, even though those situations can be encountered by the backends when running the code generated from an ABS model. For example, in the Maude semantics, a *division by zero* does not allow further reduction of that process, which may go unnoticed in the overall system, or lead to a deadlock when other processes wait on the object. In the Java backend, the underlying Java runtime will generate a Java exception through the primitive math operations, which will terminate the current (ABS) process, and lead to similar effects as in Maude.

3.1 Design Considerations

Invariants and the system. On abrupt termination of a computation, we need to establish which reaction would be required. In a distributed, loosely coupled system, a local error should not affect the complete system. So clearly here the guiding point must be that we have to keep the effects *local*. In our actor-based setting, we can take the locality even further: Although a computation failed, we can limit the effects to the *current process*. The object may still be able to process pending and future requests (although the caller of the failing process needs to be notified). But what should be the basis for further executions within this object?

The underlying motivation for the explicit release points in the language are of course the class invariants that developers rely on when designing their programs. As such, each method call expects that its respective object invariant holds upon entry (and upon awakening). This is clearly not the case under abrupt termination, before which the fields of the object may have been arbitrarily manipulated—the next release point may not have been reached.

Error handling in an object system. The mechanism we propose, defines the behavior in case of an error:

– Propagate errors through futures. The caller receives an error when reading the future.
– Default to having no explicit error handling, in which case a *process* is terminated, yet the *object* stays alive.
– Revert any partial state modifications to the current object up to the last release point.

These concepts are introduced by extending futures to propagate a possible error in the callee to the caller, providing a method to detect and handle an error contained in a future, and to terminate the caller in the case an error in a future is accessed by the default mechanism.

Linguistic support for error handling. We consider the following linguistic support to enable the envisaged error handling:

– a notion of user-defined error types
– a generalization of futures to either return values or propagate errors
– a statement **abort** `e`, which raises an error `e` and terminates the process
– a statement `f.`**safeget**, which can receive errors and values from a future `f`
– a statement **die**, which terminates the current object and all its processes

The occurrence of an error is represented in the model by means of the statement **abort** `e`, where `e` is an user defined error. These errors are represented by a special data type (see [15] for an extensive discussion of the potential design decisions). Such an abort can either be explicit in the model or can occur implicit either in internals of the execution, to represent distribution, system (e.g. out of memory) or runtime (e.g. division through zero) errors.

The semantic interpretation is dependent on the kind of ABS process the evaluation occurs in:

Active Object processes, represent the object's implicit execution of its `run` method. If in that process an **abort** `e` statement is evaluated, all current asynchronous calls to this object will abort with the error `e` and the references to this object will become invalid. Further synchronous or asynchronous calls to this object are equivalent to an **abort** `DeadObject` on the caller side. This mechanism was chosen, as the object behavior (its `run` method) is seen as an integral part of its correctness, and like an invalid state also an invalid termination of this behavior leads to an inconsistent object and therefore the object cannot be further used.

Asynchronous Call processes evaluated a method call in the called object. An **abort** `e` statement will terminate the process and return the error *e* to the associated future. Moreover, the callee will perform a rollback (see below).

Main Process. The main process (similar to Java's `main`-method entry point) represents the begin of the execution, and an **abort** there will, by convention, lead to the runtime system being terminated (in principle, this could be handled uniformly like the normal case, but in practice we prefer termination).

An automatic rollback discards all changes to the object's values since the last scheduling point, which can be either an **await** or **suspend**. This guarantees that objects only evolve from one state at a scheduling point to another, and not leave in case of an error an object in a state, which could violate the object invariant.

Extending futures to contain either the computed value or a potential error raised either by an **abort** on the callee side (or from the runtime in a distributed setting), enables error propagation over invocations. Following this, also the semantics of the `Future.get` statement needs to be adjusted: a **get** will, in presence of an error `e` in the future, lead to an implicit **abort** `e` on the caller side.

The newly introduced `Future.`**safeget** stops this propagation and allows one to react on errors. **safeget** returns a value of the algebraic data type `Result<T>`, which is defined as `Result<T> = Value(T val)|Error(String s)`. In case the future contains an error `e`, the same is returned, otherwise the constructor `Value(T v)` wraps the result value `v`. Note that due to the lack of subtyping in the type system, currently the only way to communicate an error indication is through a value of type `String`, as we cannot define a common type for all possible (incl. user-defined) errors.

The **die** `e` *statement* allows in asynchronous calls to terminate the active object. Its semantic meaning is the same as an **abort** `e` in the execution context of an active object process or init block. In other words, all pending asynchronous calls and the active object's process are terminated. This statement allows to implement linking (see below), and can be used in distributed models to simulate a disconnect from an object.

Discussion. We come back to the banking example in Listing 2 to illustrate the point of rollbacks. The general contract is that the list of `transactions` should accurately reflect the current total in the account. As the body of `withdraw` needs to modify two fields, we clearly benefit from ABS's semantics of explicit release points which guarantees that only one process is executing within the object (e.g. in Java, we would be required to explicitly declare the method as `synchronized` to achieve the same effect).

Nonetheless, even though if only by construction of the example, an **abort** would leave the object in an undesired state, as after the modification of the list of transactions the balance is no longer in sync with the banking transaction history. If an **abort** would simply terminate execution of the current process, and start processing another pending call on the current state of the object, we would observe invalid results. But with the rollback before processing another call, this assumption can easily be re-established.

Note that the ABS methodology is only concerned with *object* invariants, and this mechanism does not give us *totality* in the sense that a method either completes successfully or not at all: a rollback will not undo changes in other objects that have (transitively) occurred as the result of method calls during execution of the current process, unlike e.g. in work on a higher-order π-calculus [16]. This means on the one hand that the developer still has to actively take into account the workings of error recovery when designing the system, but on the other hand allows us to implement this feature efficiently by only keeping track of fields in the current object that are actually touched.

Compared to traditional object-oriented programming, we note that this implicit error handling strategy frees the developers from restoring state explicitly in an exception handler. However, through the **safeget** mechanism, they still have this option open.

3.2 A Practical Application of Error Propagation: Process Linking

The previously presented primitives enable an implementation of Erlang-style linking between two objects in ABS. These links are part of the foundation for Erlang's well known and successful error handling [1]. Erlang's communication model is even more loosely coupled than ABS, in that it is based on asynchronous message passing. As such, there are no method calls or explicit returns, but rather the callee has to send back a response, which will be queued in the recipient until extracted from the mailbox. Thus, a failure in the recipient process will either go unnoticed if no response messages are used or otherwise lead to an expected message not being sent/received, and in turn a corresponding potential blockage can occur in the initial sender.

Erlang's links enable mutual observation of processes. A process can link itself to another process. If one of the two processes terminates, the runtime environment sends an *EXIT* message to the other process, which contains an exit reason. Unless this exit reason is *normal* (termination because the process reached the end of the function), the linked process will terminate as well, and in consequence propagate its own *EXIT* message to its linked processes. With this *error propagation*, it is possible to let groups of processes up to the whole system terminate automatically and clean up components consisting of multiple processes.

To enable processes to observe exit messages or react on them, a process can be marked to be a system process with the *trap_exit* process flag. Such processes will not terminate when receiving an *EXIT* message, but can retrieve this message from their inbox.

Implementation in the concurrent object model. The implementation idea is to represent a link by two asynchronous calls, one to each of the objects. Each call will only terminate upon termination of the object, and thus enables the caller to take an action.

In Figure 3 a sample implementation is shown, which assumes that each class implements code similar to the `Linkable` class. A link can be established by creating a new object of class `Link`, where the link gets initialized with references to both objects (referred to as `s` and `f`), and then calling `setup` on this new link. The `setup` method will initiate the calls between the objects, by calling `waitOn` and then wait until both calls are processed, where finished calls can be seen by the counter `done`.

The `waitOn` method implemented in the `Linkable` class places the normally non-terminating asynchronous call in line 3 to the other `Linkable` it should link to. The non-termination is achieved by a simple **await** `false`, as can be seen in the `wait` method. After those calls are made, the `waitOn` method reports back to the `Link` that it succeeded, and will afterwards await the termination of the call in line 3. The only possibility for a call to `wait` to return is when the object dies. Should now this future ever contain a value it must be an error, where in line 7 we can now take an action in case that the other object terminated, which will be in the default case a subsequent termination of the local object, by executing **die** `e`.

Linking in a producer consumer environment can be used to bind both objects together, so that a termination of the producer or consumer leads to the termination of the other party as well. In Figure 4 we see a `Producer` and `Consumer`, modeled as ABS classes, where the `Producer` sends a new input to the `Consumer` via an asynchronous call. Both classes have to implement the `Linkable` interface and include the shown default implementation of `wait` and `waitOn`.

Setting up a Link between `Producer` and `Consumer` is performed by the first two lines in the `Producer`'s `run` method. We construct the `Link` object and

```
class Link(Linkable f,Linkable s)
  implements Link{
  Int done=0;
  Unit setup(){
   f!waitOn(this,s);
   s!waitOn(this,f);
   await done==2;
  }

  Unit done(){
   done=done+1;
  }
}
```

```
class Linkable() implements Linkable{
  Unit waitOn(Link l,Linkable la){
    Fut<Unit> fut=la!wait();
    l!done();
    await fut?;
    case fut.safeget {
     Error(e) => die e;
    }
 }
 Unit wait(){
   await false;
 }
}
```

Fig. 3. Implementation of links in ABS

```
class Producer(Consumer c)
  implements Linkable{
 Unit run(){
  Link lConsumer= new Link(this,c);
  await lConsumer!setup();
  // produce
  c!consume(X);
 }
 // include wait and waitOn
}
```

```
class Consumer()
  implements Linkable{

 Unit consume(String x){
    // consume
 }

 // include wait and waitOn,
}
```

Fig. 4. Links between a Producer and a Consumer

initialize the link via the `setup` method. A more detailed view of asynchronous calls and their lifetime is presented in the sequence diagram in Figure 5, arrows represent an invocation and a possible return value, and boxes represent the duration of the call on the callee side. First, the link is setup, two inputs are produced, and after that the `Consumer` aborts, which also terminates the `Producer`.

Before the `consume` calls, all necessary invocations to establish the `wait` calls, which can be seen as a monitor if the object is still alive, are shown. After that we see that two inputs from the `Producer` are sent to the `Consumer`, where the `wait` calls are still pending. In the end, the `Consumer`, and in consequence also the `wait` call, terminate. The termination leads to the retrieval of the exit reason (in form of an error) by the `Producer` from the associated future, which results in its termination as well.

One of the current limitations of this design is that due to the lack of subclassing, the boiler-plate implementation of the methods `wait` and `waitOn` in any class needs to be replicated (such as `Producer` and `Consumer` above). ABS offers so-called *deltas* to support assembly of *software product lines*. Although this feature can be used here in principle to inject code into a class, according to the current syntax of deltas, the method bodies would still have to be replicated in *each* delta. A potential improvement would be an extension of ABS which would allow injecting code into all classes implementing a particular interface.

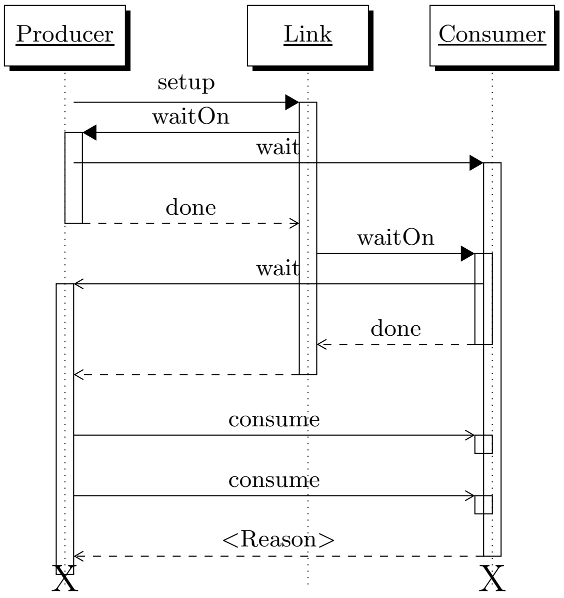


Fig. 5. Asynchronous calls in the Producer-Consumer example

Such functionality is well-known in *aspect-oriented programming*, and the ABS compiler should be easy to extend with a similar feature.

4 Operational Semantics and Application

A complete operational semantics of the core ABS language can be found in [11]. This section presents an operational semantics of the new language elements discussed in Section 3, omitting or simplifying parts that are not necessary for understanding the new error model. Figure 6 presents the runtime syntax of the language, while Figure 7 contains the operational semantics rules for the new rollback behavior, **abort** and **die** statements, and error propagation via futures.

The runtime state is a collection cn of objects, futures and method invocations. Objects are denoted $o(a, a', p, q)$, where a is the object state, a' the safe state at the previous suspension point. Dead objects are represented by their identifier o only. Object and process states a are mappings from identifiers to values, p is the currently running process or the symbol **idle** (denoting an object not currently running any process), and q is the process queue. A process p is written as $\{a|s\}$ with a a mapping from local variable identifiers to values and s a statement list.

In Figure 7 we elide the step of reducing expressions to values – evaluation is standard and can be seen in [11]. The Suspend rule saves the current state a, while the Abort rule reinstates a saved state while also removing the current process and filling the future f with an error term. The Die rule deactivates the object and fills all futures of the object's processes with an error term. The

$$
\begin{array}{rcl}
\mathit{cn} & ::= & \epsilon \mid \mathit{fut} \mid \mathit{object} \mid \mathit{invoc} \mid \mathit{cn}\ \mathit{cn} \\
\mathit{fut} & ::= & f \mid f(\mathit{val}) \\
\mathit{object} & ::= & o(a, a', p, q) \mid o \\
\mathit{process} & ::= & \{a \mid s\} \\
q & ::= & \epsilon \mid \mathit{process} \mid q\ q \\
\mathit{invoc} & ::= & m(o, f, \overline{v})
\end{array}
\qquad
\begin{array}{rcl}
a & ::= & T\ x\ v \mid a, a \\
p & ::= & \mathit{process} \mid \textbf{idle} \\
\mathit{val} & ::= & v \mid \mathit{error} \\
v & ::= & o \mid f \mid \mathit{data} \\
\mathit{error} & ::= & e(\mathit{val})
\end{array}
$$

Fig. 6. Runtime syntax. Overall program state is a set *cn* of futures, objects and invocation messages. Literals v are object identifiers o, future identifiers f, and number and string literals *data*.

Dead-Call rule provides a default error term as the result of a method call to a dead object. The other rules show the behavior of normal execution for these cases.

4.1 Discussion

From an implementation perspective, we note that the rollback mechanism appears reasonably cheap, as only that part of the state of the current object needs to be duplicated which is actually modified. This is easy to implement since ABS does not have destructive modification of data structures.

How to make best use of the rollback-mechanism is still up to the developer. We note that compared to traditional exception handling, a single method essentially corresponds to a **try**-block, whereas the caller specifies through a **safeget** and a subsequent case-distinction the possible **catch**-blocks, or decides to propagate any exceptions through **get**.

4.2 Application: Supervision

In Erlang the idea to let processes observe each other was taken further by constructing trees, where so called *supervisors start, observe and restart* their child processes. Supervision is one of the very important concepts, which is part of Erlang's highly regarded error handling capabilities [19]. Plugging in a supervisor as child of another supervisor generates a tree structure, which describes a structural view on components of a system. This tree structure enables both restarting of faulty leaves and of larger subtrees in case of repeated errors in a subsystem. So a faulty system with a supervisor tries to restart larger and larger parts of the whole system until enough faulty state is discarded and it is able to continue its operation.

Supervision for concurrent objects. Through linking, we can now apply the concept of supervision to concurrent objects. This enables modeling of a statically typed supervision tree that maintains active objects.

To achieve a very generalized supervisor implementation we want to separate it from the concrete way of starting and linking children and want to be able to

$$\textsc{(Suspend)}\quad \begin{array}{c} o(a, a', \{l \mid \textsf{suspend}; s\}, q) \\ \to o(a, a, \textsf{idle}, \{l \mid s\} \circ q) \end{array}$$

$$\textsc{(Activate)}\quad \frac{p = \mathit{select}(q, a, cn)}{\begin{array}{c} o(a, a', \textsf{idle}, q)\ cn \\ \to o(a, a', p, (q \setminus p))\ cn \end{array}}$$

$$\textsc{(Await-Incomplete)}\quad \begin{array}{c} o(a, a', \{l \mid \textsf{await}\ f?; s\}, q)\ f \\ \to o(a, a', \{l \mid \textsf{suspend}; \textsf{await}\ f?; s\}, q)\ f \end{array}$$

$$\textsc{(Await-Complete)}\quad \begin{array}{c} o(a, a', \{l \mid \textsf{await}\ f?; s\}, q)\ f(\mathit{val}) \\ \to o(a, a', \{l \mid s\}, q)\ f(\mathit{val}) \end{array}$$

$$\textsc{(Return)}\quad \frac{f = l(\textsf{destiny})}{\begin{array}{c} o(a, a', \{l \mid \textsf{return}(v); s\}, q)\ f \\ \to o(a, a, \textsf{idle}, q)\ f(v) \end{array}}$$

$$\textsc{(Abort)}\quad \frac{f = l(\textsf{destiny})}{\begin{array}{c} o(a, a', \{l \mid \textsf{abort}(v); s\}, q)\ f \\ \to o(a', a', \textsf{idle}, q)\ f(e(v)) \end{array}}$$

$$\textsc{(Async-Call)}\quad \frac{\mathit{fresh}(f)}{\begin{array}{c} o(a, a', \{l \mid x = o'!m(\overline{v}); s\}, q) \\ \to o(a, a', \{l \mid x = f; s\}, q)\ m(o', f, \overline{v})\ f \end{array}}$$

$$\textsc{(Bind-Mtd)}\quad \frac{p' = \mathit{bind}(m, o, \bar{v}, f)}{\begin{array}{c} o(a, a', p, q)\ m(o, f, \bar{v}) \\ \to o(a, a', p, p' \circ q) \end{array}}$$

$$\textsc{(Die)}\quad \frac{f = l(\textsf{destiny})\quad cn' = \mathit{abort\text{-}futures}(cn, q, v)}{\begin{array}{c} o(a, a', \{l \mid \textsf{die}(v); s\}, q)\ f\ cn \\ \to o\ f(e(v))\ cn' \end{array}}$$

$$\textsc{(Dead-Call)}\quad \begin{array}{c} o\ f\ m(o, f, \bar{v}) \\ \to o\ f(e(\texttt{"dead object"})) \end{array}$$

$$\textsc{(Read-Fut)}\quad \begin{array}{c} o(a, a', \{l \mid x = f.\textsf{get}; s\}, q)\ f(v) \\ \to o(a, a', \{l \mid x = v; s\}, q)\ f(v) \end{array}$$

$$\textsc{(Read-Fut-Error)}\quad \begin{array}{c} o(a, a', \{l \mid x = f.\textsf{get}; s\}, q)\ f(e(v)) \\ \to o(a, a', \{l \mid \textsf{abort}(v); s\}, q)\ f(e(v)) \end{array}$$

$$\textsc{(Safe-Read)}\quad \begin{array}{c} o(a, a', \{l \mid x = f.\textsf{safeget}; s\}, q)\ f(\mathit{val}) \\ \to o(a, a', \{l \mid x = \mathit{val}; s\}, q)\ f(\mathit{val}) \end{array}$$

Fig. 7. Operational semantics. The following helper functions are assumed: *bind* creates a new process given a method name m, object o, arguments $\overline{v}$ and future f; *abort-futures* transforms a configuration, filling all futures f referenced from processes in queue q with an error term $e(v)$ while returning all other parts of the configuration unchanged; *select* chooses a process from a queue q that is ready to run.

```
Unit start(SupervisibleStarter child){
 SupervisorLink sl=
      new SupervisorLink(this,child);
 Link l=new Link(sl,this);
 await l!setup();
 links=Cons(sl,links);
 sl.start();
}
```

(a) Start a child

```
Unit died(SupervisibleStarter ss,
         String error){
 case strategy {
  RestartAll => this.restart();
  RestartOne => this.start(ss);
  Prop => die error;
 }
}
```

(b) Handle a deceased child

Fig. 8. Key methods of the `Supervisor`

define different restart strategies. These strategies define the actions taken if a child terminates. Therefore we implemented a class `Supervisor` with following parameters: a list of `SupervisorStarter` objects, each of which specifies one child and implements the start and linking of this child; a strategy, which can be one of the following:

Restart one: Only the terminated child is restarted.
Restart all: If a child dies, it and all its siblings will be restarted.
Propagate: The supervisor and all children will terminate and the error will be thereby propagated to the next supervisor, ending at the root node of the runtime system.

This can be easily extended with other interesting strategies like rate limiting, e.g. propagating an error if a certain frequency of crashes is exceeded.

The implementation of the supervisor requires special considerations, as a supervisor has to start a list of children, keep track of them, has to detect a link failure and be able to forcefully terminate a child (for the *restart all* strategy). As the standard implementation of the link mechanism, shown in Figure 3, has on the error receiving side no indication about the source of the link error, every link to a child is represented by an object of class `SupervisorLink`. This object keeps the reference of the child specification (the `SupervisibleStarter` object) and passes it along to the `Supervisor`'s `died` method, which is depicted in Figure 8b. Furthermore this design allows one to forcefully kill one child, by terminating the associated `SupervisorLink`, which will—via linking—terminate the child.

For starting a child, a new `SupervisorLink` has to be created and linked to, so that in case the supervisor itself terminates (e.g. when the strategy is to propagate) all `SupervisorLink`s and children are terminated as well. This method is shown in Figure 8a.

5 Conclusion and Related Work

We have presented an extension to a concurrent object language, which incorporates automatic rollback to a "safe" (as conceptually defined by the developer through a class invariant) state for the object that encountered an *abort*. Aborts either occur in the form of runtime errors, through an explicit call similarly to **throw**ing an exception, or from accessing a future which holds the result of an aborted computation.

The propagation- and detection mechanism for such faults allows us to model Erlang-like process linking, and the *safe* way of accessing futures corresponds roughly to exception handling with a distinction on the return result (normal return value vs. fault plus description).

We have implemented the proposed extension in a straight-forward manner in the prototypical (non-distributed) Erlang backend for ABS, and in the Maude simulator: The sources are publicly available in the ENVISAGE git repository at `http://envisage-project.eu`.

Related Work. Asynchronous computation with *futures* has been standardized in the Java API since Java SE 5 [10]. Due to the limitations of the so-called *generics* in the type system, no subtyping on futures is possible: this leads to the situation that (synchronous) method calls may make use of covariant return types, but for a type `B extending A`, a **Fut**`<B>` cannot be assigned to a **Fut**`<A>`. Our futures, based on ABS, do not have this limitation as futures stem from the functional data types and thus subtyping over parameterized types is safe due to the lack of destructive updates/writes. As first-class citizens, the ABS futures do not offer any cancellation and a process cannot affect another process except through sending messages (the Java API offers advisory cancellation, and—discouraged—forceful termination of threads).

Compared to Java futures, the ABS futures are intended to scale massively: while due to the limitations in Java's thread model only a restricted (by memory/stack requirements) number of threads can be effectively active (the standard reference [10] gives a limit in the "few thousands or tens of thousands", usually scheduled by an execution service); their intended use in ABS clearly follows Erlang's notion of virtually unbounded, light-weight, disposable threads.

A related failure model for an ABS-like language has also been discussed in [12]. To enable coordinated rollbacks, *compensations* are attached to method *returns*, in case a later condition indicates that a rollback across method calls should be necessary. The authors illustrate however that the distributed nature of compensation still does not make it easier to maintain *distributed invariants* involving several objects. Rollbacks in a concurrent system and their intricacies have also been discussed in the context of a higher-order π-calculus by Lanese et al. [16]. The entire design space of fault handling in a loosely coupled system is discussed in [15], but focuses on a more traditional approach of exception handlers to give developers an explicit means of recovery, instead of the implicit rollbacks presented here.

Unlike Java Card's transactions [6] our extension does not allow selective *non-atomic* updates, where a persistent value is modified within a transaction and *not* rolled back with the transaction. Our implementations do not store the entire heap upon method activation, but only the state of the current object.

A corresponding proof-theory as developed by Mostowski [17] for JAVA CARD-support in the KeY system should likewise be feasible for our approach.

Future work: The current rollback mechanism should also be easy to extend to *transactions* through a combination of versioning the object state and speculative execution. Also, rollbacks for a group of objects should at least semantically be easy to model, yet maintaining object graphs as additional state may make this approach too costly: every method invocation on another object would make this object a member of the transaction, and all objects would have to reach release points simultaneously to commit. Additionally, a distributed implementation of checking for such a commit would most likely be prohibitive. Instead of arbitrary object groups derived again following the discussion in [15], one may instead take advantage of so-called *concurrent object groups* (which are already present in ABS, but not discussed in this paper). They are used in ABS to model groups of objects running e.g. on the same node or hardware. Because of the intentionally tight coupling, one consideration is that a **die**-statement may even have the consequence of terminating the processes of an entire group, instead of the limited effect on the local object only that we discussed here.

Although the asynchronous communication mechanism together with the introduced failure mechanisms allows us to describe the communication behavior in a distributed system, the current semantics treats all calls—whether remote or local—the same. While this location transparency is also a feature of the Erlang language, it would be useful to reflect the topology of the system and resource aspects (such as processing power and communication latency) of the different nodes in a model. To this end, in [14] *deployment components* were introduced, which give the modeler the possibility to specify where objects are created and consequently where their processes run. Note that in contrast to Erlang, *objects* are allocated at creation-time, whereas Erlang allocates processes. On top of deployment components, resource costs and capabilities can be modeled and execution times can be estimated under different resource and deployment models. Simulation can then be used to examine the behavior of the (distributed) system wrt. artificially injected faults and deadline misses.

With respect to the supervision trees, we note that in the Erlang community, since the tree structure is specified through code, there was an interest in reverse-engineering the actual hierarchy for purposes of static analysis from the source code [18]. We hope that for top-down development, specification of the hierarchy can be made independent of the code, and is conversely more amenable to verification.

References

1. Armstrong, J.: Erlang—a survey of the language and its industrial applications. In: Proc. INAP, vol. 96 (1996)
2. Armstrong, J.: Programming Erlang: Software for a Concurrent World. Pragmatic Bookshelf (2007)

3. Bjørk, J., de Boer, F.S., Johnsen, E.B., Schlatte, R., Tapia Tarifa, S.L.: User-defined schedulers for real-time concurrent objects. Innovations in Systems and Software Engineering 9(1), 29–43 (2013)
4. Candea, G., Fox, A.: Crash-only software. In: Jones, M.B. (ed.) HotOS, pp. 67–72. USENIX (2003)
5. Caromel, D., Henrio, L.: A Theory of Distributed Objects. Springer (2005)
6. Chen, Z.: Java Card Technology for Smart Cards. Addison-Wesley (2000)
7. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.L.: All About Maude - A High-Performance Logical Framework. LNCS, vol. 4350. Springer, Heidelberg (2007)
8. de Boer, F.S., Clarke, D., Johnsen, E.B.: A complete guide to the future. In: De Nicola, R. (ed.) ESOP 2007. LNCS, vol. 4421, pp. 316–330. Springer, Heidelberg (2007)
9. Din, C.C., Dovland, J., Johnsen, E.B., Owe, O.: Observable behavior of distributed systems: Component reasoning for concurrent objects. Journal of Logic and Algebraic Programming 81(3), 227–256 (2012)
10. Goetz, B., Peierls, T., Bloch, J., Bowbeer, J., Holmes, D., Lea, D.: Java Concurrency in Practice. Addison-Wesley (2006)
11. Johnsen, E.B., Hähnle, R., Schäfer, J., Schlatte, R., Steffen, M.: ABS: A core language for abstract behavioral specification. In: Aichernig, B.K., de Boer, F.S., Bonsangue, M.M. (eds.) FMCO 2010. LNCS, vol. 6957, pp. 142–164. Springer, Heidelberg (2011)
12. Johnsen, E.B., Lanese, I., Zavattaro, G.: Fault in the future. In: De Meuter, W., Roman, G.-C. (eds.) COORDINATION 2011. LNCS, vol. 6721, pp. 1–15. Springer, Heidelberg (2011)
13. Johnsen, E.B., Owe, O.: An asynchronous communication model for distributed concurrent objects. Software and Systems Modeling 6(1), 35–58 (2007)
14. Johnsen, E.B., Schlatte, R., Tapia Tarifa, S.L.: Modeling application-level management of virtualized resources in ABS. In: Beckert, B., Damiani, F., de Boer, F.S., Bonsangue, M.M. (eds.) FMCO 2011. LNCS, vol. 7542, pp. 89–108. Springer, Heidelberg (2013)
15. Lanese, I., Lienhardt, M., Bravetti, M., Johnsen, E.B., Schlatte, R., Stolz, V., Zavattaro, G.: Fault model design space for cooperative concurrency. In: Margaria, T., Steffen, B. (eds.) ISoLA 2014, Part II. LNCS, vol. 8803, pp. 22–36. Springer, Heidelberg (2014)
16. Lanese, I., Mezzina, C.A., Schmitt, A., Stefani, J.-B.: Controlling reversibility in higher-order Pi. In: Katoen, J.-P., König, B. (eds.) CONCUR 2011. LNCS, vol. 6901, pp. 297–311. Springer, Heidelberg (2011)
17. Mostowski, W.: Formal reasoning about non-atomic Java Card methods in dynamic logic. In: Misra, J., Nipkow, T., Sekerinski, E. (eds.) FM 2006. LNCS, vol. 4085, pp. 444–459. Springer, Heidelberg (2006)
18. Nyström, J., Jonsson, B.: Extracting the process structure of Erlang applications. In: Erlang Workshop, Florence, Italy (September 2002), `http://www.erlang.org/workshop/nystrom.ps`
19. Vinoski, S.: Reliability with Erlang. IEEE Internet Computing 11(6), 79–81 (2007)

Fault Model Design Space for Cooperative Concurrency*

Ivan Lanese[1], Michael Lienhardt[1], Mario Bravetti[1], Einar Broch Johnsen[2], Rudolf Schlatte[2], Volker Stolz[2], and Gianluigi Zavattaro[1]

[1] Focus Team, Università di Bologna/INRIA, Italy
`{lanese,lienhard,bravetti,zavattar}@cs.unibo.it`
[2] Department of Informatics, University of Oslo, Norway
`{einarj,rudi,stolz}@ifi.uio.no`

Abstract. This paper critically discusses the different choices that have to be made when defining a fault model for an object-oriented programming language. We consider in particular the ABS language, and analyze the interplay between the fault model and the main features of ABS, namely the cooperative concurrency model, based on asynchronous method invocations whose return results via futures, and its emphasis on static analysis based on invariants.

1 Introduction

General-purpose modeling languages exploit *abstraction* to reduce complexity [20]: modeling is the act of describing a system succinctly by leaving out some aspects of its behavior or structure. Software models primarily focus on the functional behavior and the logical composition of the software. Modeling formalisms can have varying levels of detail and can express structural properties (for example UML diagrams), interactions (π-calculus), or the effects of functions or methods (pre- and post-conditions), etc.

Concurrent and distributed systems demand flexible communication forms between distributed processes. While object-orientation is a natural paradigm for distributed systems [15], the tight coupling between objects traditionally enforced by method calls may be criticized. Concurrent (or active) objects have been proposed as an approach to concurrency that blends naturally with object-oriented programming [1, 22, 32]. Several slightly differently flavored concurrent object systems exist for, e.g., Java [3, 30], Eiffel [5, 26], and C++ [25]. Concurrent objects are reminiscent of Actors [1] and Erlang processes [2]: objects are inherently concurrent, conceptually each object has a dedicated processor, and there is at most one activity in an object at any time. Thus, concurrent objects encapsulate not only their state and methods, but also a single (active) thread of control. In the concurrent object model, *asynchronous method calls* may be used to better combine object-orientation with distributed programming by reducing the temporal coupling between the caller and callee of a method, compared

* Partly funded by the EU project FP7-610582 ENVISAGE.

to the tightly synchronized (remote) method invocation model (of, e.g., Java RMI [27]). Intuitively, asynchronous method calls spawn activities in objects without blocking execution in the caller. Return values from asynchronous calls are managed by *futures* [14,23,32]. Asynchronous method calls and futures have been integrated with, e.g., Java [11,19] and Scala [13] and offer a large degree of potential concurrency for deployment on multi-core or distributed architectures.

ABS is a modeling language targeting distributed systems [17]; the language combines concurrent objects and asynchronous method calls with *cooperative scheduling* of method invocations. In ABS the basic unit of computation is the *concurrent object group* (cog): a cog provides to a group of objects a shared processor. Method invocations on an object of a cog instantiate a new task that requires the cog's processor in order to execute. Cooperative scheduling allows tasks to suspend in a controlled way at explicit points in the code, so that other tasks of the object can execute. The **suspend** and **await** commands are used to explicitly release the processor: the difference between the two commands is that **await** has an associated boolean guard expressing under which condition the task should be re-activated by the scheduler. Asynchronous method invocations are used among objects belonging to different cogs; at each asynchronous method invocation a *future* is instantiated to store the return value. Futures are first class citizens in ABS and are accessed via a **get** command; **get** is blocking because a task, executing **get** on a future of a method invocation which has not yet completed, blocks and keeps the processor until the future is written. To avoid keeping the processor, one can use an **await** `f?` to ensure that future `f` contains a value.

ABS has a formal, executable semantics; ABS models can be run on a variety of backends and can be verified using the KeY proof checker [4]. In particular, asynchronous method calls and cooperative scheduling allow the verification of distributed and concurrent programs by means of sequential reasoning [8]. In ABS this is reflected in a proof system for local reasoning about objects where the class invariant must hold at all scheduling points [9]. Although ABS targets distributed systems, a notable abstraction of the language design is that faults are currently not considered part of the behavior to be modeled. On the other hand, dealing with faults is an essential and notoriously difficult part of developing a distributed system; this difficulty is exacerbated by the lack of clear structuring concepts [7]. A well-designed model is essential to understand potential faults and reason about robustness, especially in distributed settings. Thus, it is interesting to extend a modeling language such as ABS in order to model faults and how these can be resolved during the system design.

It is common in the literature to distinguish errors due to the software design (sometimes called *faults*) from random errors due to hardware (sometimes called *failures*). For software deployed on a single machine, such hardware failures entail a crash of the program. A characteristic of distributed systems is that failures may be *partial* [31]; i.e., the failure may cause a node to crash or a link to be broken while the rest of the system continues to operate. In our setting, a strict separation between faults and failures may seem contrived, and we will refer to unintended behavior caused by both the software and hardware as faults. A fault is *masked* if the fault is not detected by the client of the service in which the fault occurs. In hierarchical fault models, faults can propagate along the path of service requests; i.e., a fault at the server level can result in a (possibly different) fault at the client level. In a synchronous communication model, a client object can only send one method call at the time whereas in an asynchronous communication model, the client may spawn several calls. Thus, it need not be clear for a client object which of the spawned calls resulted in a specific fault in the asynchronous case. However, asynchronous method calls in ABS allow results to be *shared* before they are returned: futures are first-class citizens of ABS and may be passed around. First-class futures give rise to very flexible patterns of synchronization, but they further obfuscate the path of service requests and thus of fault propagation.

This paper discusses an extension of the semantics of the ABS modeling language to incorporate a robust fault model that is both amenable to formal analysis and familiar to the working programmer. The paper considers how faults can be introduced into ABS in a way which is faithful to its syntax, semantics, and proof system, and discusses the appropriate introduction of faults along three dimensions: fault representation (Section 2), fault behavior (Section 3), and fault propagation (Section 4).

2 How Are Faults Represented?

Exceptions are the language entities corresponding to faults in an ABS program's execution. ABS includes two kinds of entities which in principle can be used to represent faults: *objects* and *datatypes* (datatypes [16] are part of the functional layer of ABS, and abstract simple, common structures like lists and sets).

Exceptions as Objects. Representing exceptions as objects allows for a very flexible management of faults. Indeed, in this setting exceptions would have both a mutable state and a behavior. Also, one could define new kinds of exceptions using the interface hierarchy. Finally, exceptions would have identities allowing to distinguish different instances of the same fault. However, most of these features are not needed for faults: faults are generated and consumed, but they are static and with no behavior. Representing them as objects would allow a programming style not matching the intuition and difficult to understand. Furthermore, in ABS static verification is a main concern, and semantic clarity is more needed than in other languages. For this reason we think that in the setting of ABS exceptions should not be objects.

Exceptions as Datatypes. Datatypes fit more with the intuition of exceptions as described before: they are simple values with no identity nor behavior. However, in ABS datatypes are closed, meaning that once a datatype has been declared, it is not possible to extend it with new constructors. This is a potential problem in using them to represent exceptions. Indeed, we would like the datatype for exceptions to include system-defined exceptions such as Division by Zero or Array out of Bound, and to be extended to accommodate user-defined exceptions. Also, for modularity reasons, programmers of an ABS module should be able to declare their own exceptions, thus exception declaration cannot be centralized. User-defined exceptions are not only handy for the programmer, but may also help the definition of invariants by tracking the occurrence of specific conditions. We discuss below a few possible design choices related to the definition of user-defined exceptions.

Allow open datatypes in ABS. In this setting exception would be an open datatype [24], and other ABS datatypes could be open as well. The declaration of system-defined exceptions can be done as:

```
open data Exception = NullPointerException
                    | RemoteAccessException
```

where the keyword **open** specifies that the datatype is open (in principle open and closed datatypes may coexist). Then one can add user-defined exceptions as:

```
open data Exception = ... | MyUserDefinedException
```

However, this is a major modification of datatypes, a key component of ABS, and introducing this additional complexity only to accommodate exceptions may not be a good choice. In fact, handling open datatypes is in contrast with the fact that ABS type system is nominal. One would need to resort to a structural type system (similar to, e.g., OCaml's variants [29]) to ensure that a pattern matching is complete, which is far less natural.

Allow any datatype to be an exception. In this setting any value of any datatype may be used as an exception (the fact of declaring which datatypes are actually used as exceptions does not change too much the setting). User-defined datatypes can be added by simply defining new datatypes. When the programmer wants to catch an exception, he has to specify which types of exceptions he can catch, and do a pattern matching both on the type and on its constructor to understand which particular fault happened. This produces a syntax like:

```
try { ... }
catch(List e) {
  case(e) {
   | Empty => ...
   | Cons(v,e2) => ...
} }
catch(NullPointerException e) { ... }
catch(_ e) { ... /* capture all exceptions */ }
```

where a special syntax _ is needed to catch exceptions of any type, since there is no hierarchy for datatypes in ABS. Note that in case the exception has type `List` a **case** is done to analyze its structure. A difficulty in applying this approach to ABS is due to the fact that in ABS values do not carry their type at runtime, but adding such an information seems not to have relevant drawbacks.

Exceptions as a new kind of value. In this setting exceptions are a separate kind of value, at the same level of objects and datatypes. The type Exception is open. New exceptions (both system- and user-defined) can be declared as follows:

```
exception NullPointerException
exception RemoteAccessException
...
exception MyUserDefinedException(Int, String)
```

Pattern matching can be used to distinguish different exceptions:

```
try { ... }
catch(e) {
  NullPointerException => ...
  MyUserDefinedException(n,s) => ...
  e2 => ...
  ...
}
```

Structural typing can be used if one wants to check that all exceptions possibly raised are caught, as, e.g., in Java.

Discussion. The simplest approach is to model exceptions as a closed datatype. However, if open exceptions are desired to increase the expressive power, the last solution is the one with minimal impact on the existing ABS language. Allowing any datatype as an exception also seems a viable option, but with a more substantial impact on the existing structure of ABS.

3 Which Is the Behavior of Faults?

Faults interrupt the normal control flow of the program. A first issue concerning faults is how they are generated. Concerning fault management, it is a common agreement that faults are manipulated with a **try**/**catch** structure, and we do not see any reason to change this approach in our design for ABS. However, after this choice has been taken, the design space is still vast and many questions still need to be investigated.

Fault Generation. In programming languages, faults can be generated either by an explicit command such as **throw** `f` where `f` is the raised fault, or by a normal command. For instance, when evaluating the expression `x/y` a Division by Zero exception may be raised if `y` is 0. In this second case, which exception is raised is not explicit, but defined by the semantics of the command. After having been raised, the two kinds of exceptions are indistinguishable. A third kind of exception may be considered in ABS. Indeed, ABS is currently evolving towards having an explicit distribution, and in this setting localities or links may break. The only remote interaction in ABS is via asynchronous method invocation, and the corresponding **await**/**get** on the created future. In principle, network problems could be notified either during invocation, or during the **await**/**get**.

However, invocation is asynchronous, and will not check for instance whether the callee will receive and/or process the invocation message. For the same reason, it is not reasonable that it checks for network problems. Clearly, the **get** should raise the fault, since no return value is available.

The behavior of **await** depends on its intended semantics. If executing the statement **await** `f?` means that the process whose result will be stored in `f` has successfully finished, then the **await** needs to synchronize with the remote computation and should raise a pre-defined fault upon timeout and network errors. In this setting thus network faults are raised by both **await** and **get**. On the other hand, if executing **await** `f?` gives only the guarantee that a subsequent `f.`**get** will not block, then all faults, including network- and timeout-related faults, can be raised by **get** exclusively.

Fault Management. As discussed in the beginning of the section, we use the common **try**/**catch** structure to manage faults. This structure sometimes also features an additional block **finally**. The **finally** block specifies some code that must be executed both if no exception is raised and if it is. A common use of the **finally** block is to release resources which need to be freed whatever the result of the computation is.

```
try { ... }
catch(MyFirstException e) { ... }
catch(MySecondeException e) {... }
finally { P }
```

For instance, `P` may close a file used inside the **try** block. The **finally** block is very convenient for programming, but may not be needed in the core language. Indeed, in many cases it can be encoded. The encoding instantiated on the example above is as follows:

```
try {
  try { ... }
  catch(MyFirstException e) { ... }
  catch(MySecondeException e) { ... }
} catch(_ e) {
  P
  throw e;
}
P
```

Essentially, one has to catch all the exceptions, do `P` and rethrow the same exception. `P` also needs to be replicated at the end, so to be executed if no exception is raised. Note that this encoding relies on always having exactly one **return** statement per method, at its end (this is the recommended style of programming in ABS), and on the ability to catch all exceptions and to be able to rethrow them identically. Actually, in principle, one can also consider some uncatchable faults, but this seems not particularly relevant in practice.

For resource management, an alternative to the **finally** block is the autorelease mechanism of Java 7 [28], which automatically releases its resource at the end of the block. Encoding such a mechanism in ABS could be done using an approach similar to the one above for the **finally** block.

Fault Effects. We have discussed how to catch faults. However, it may happen that a fault is not caught inside the method raising it. Then, as already said, the fault should interrupt the normal flow of computation, i.e. killing the running process. However, one may decide to kill a larger entity. Suitable candidates in ABS are the object where the fault has been raised, or its cog. Now, remember that in ABS there is a strong emphasis on correctness proofs based on invariants, and that whenever a process releases the lock of an object the class invariant must hold. An uncaught fault releases the lock by killing the running process. This means that whenever an uncaught fault may be raised, the invariant must hold. Since faults may be raised by many constructs, including expressions and **get**, ensuring this may be particularly difficult, and may require in practice to manage all the faults inside the method raising them. However, this is undesirable since a method may not have enough information to correctly manage a given fault. One can try to define a weaker invariant, but this may be difficult. A solution is to decide that a fault may not only kill the process, but also the object whose invariant may be no more valid. An even more drastic solution is to kill the whole cog. This may be meaningful if invariants involving different objects (of the same cog) are considered. However, this kind of invariant is currently not considered in ABS, thus the introduction of mechanisms for killing a whole cog seems premature.

Effect Declaration. In classic programming languages, the only effect of an uncaught fault is to kill the running process. However, we just discussed that also killing the whole object (or cog) is a possible effect. One may want to have different effects for different faults. More in general, different faults may have different properties. Another possible property may describe whether a fault can be caught or not. Whatever the set of possible properties is, an important issue is where those properties are associated with the raised fault. One can have a keyword **deadly** specifying that a given exception will kill the whole object if uncaught, while the behavior of just killing the process can be considered the default behavior. We can see three possibilities here. Properties may be specified:

when an exception is declared: for instance, one may write

```
deadly exception NullPointerException
```

A main drawback of this approach is that the same exception will behave the same everywhere. Intuitively, an exception may be deadly for an object where the invariant cannot be restored, and not for another one where the fault has no impact on the invariant. Note also that if any datatype can be an exception, then one has to specify properties for each datatype, e.g. **deadly** `Int`. Actually, this second drawback is mitigated by choosing suitable default values for properties.

when an exception is raised: for instance, one may write

```
throw deadly NullPointerException
```

or also shorten it into **die** `NullPointerException`. Clearly, this approach is only reasonable for exceptions raised by an explicit throw (unless one wants to write something like `x=y/0` **deadly**). The approach is also less compositional, thus less suitable for static analysis. In fact, to understand the behavior of an exception it is not enough to look at declarations. For instance, the same exception may be either deadly or not for the same method, depending on how it has been raised. Note also that this approach would break the encoding of **finally** above, since there is no way to rethrow an exception with the exact same properties.

in the signature of the method raising the exception: for instance, one may write

```
Int calc(Int x) deadly: NullPointerException {...}
```

Clearly, this approach is viable only for properties relevant when the exception exits the method, such as deadly. It would not work for instance to specify whether an exception can be caught or not. Notice that this approach integrates well with the declaration of which faults a method may raise, useful to statically verify that all exceptions are caught. In fact, one could write

```
Int calc(Int x) throws: DivisionByZero,
                         deadly NullPointerException {...}
```

More in general, this approach is suitable for static analysis, since a method declaration also provides the information on the behavior of exceptions raised by the method itself. The same information is useful also for the programmer, in particular when using methods he did not write himself.

Discussion. We think that in the context of ABS, a fault may have two different effects: either killing the process or the whole object, depending on whether the object invariant holds or not. Whether a fault should kill the whole object or not should be declared at the level of method signature to enhance compositionality. Note that in the most used object-oriented languages, objects are never killed as a result of an exception: indeed such a feature is relevant in ABS because of its emphasis on analysis based on invariants, and no widespread object-oriented language has been developed according to this philosophy. A possible alternative to kill the object would be to roll back state changes. A transparent rollback [10] in our setting could lead to the last release point, where one is sure the invariant holds. However such an approach, discussed in [12], is not always satisfying. Indeed, rolling back only locally may easily lead to inconsistencies between different local states (what corresponds to break invariants concerning multiple objects). On the other hand, global rollback as in [21] results in an overly complex semantics. Furthermore, if local rollback is needed in particular cases, it can usually be encoded. Similarly, the **finally** construct is not strictly needed, since with the choices we advocate it can be encoded.

4 How Do Faults Propagate?

We have discussed in the previous section the effect of a fault on the process or object where it is raised. However, in case of fault, in particular of uncaught fault, it is reasonable to propagate the exception also to other processes/objects related to it. In particular, possible targets for propagation are processes interested in the result of the computation, processes that have been invoked by the failed one, processes in the same object/cog of the failed one, processes trying to access an object after it died.

Propagation through the Return Future. In a language with synchronous method invocation the only process that can directly access the result of the computation is the caller. However, in languages with asynchronous method invocation any process receiving the future can directly access the result of the computation. The caller may be or may not be one of them, and indeed may even terminate before the result of the computation becomes ready. Thus we discuss here notification of faults to the processes synchronizing with the future. We have two possibilities: processes may synchronize with the future either with a **get** or with an **await** statement. The case of **get** is clear: those processes are interested in the result of the computation, in case of fault no correct result is available and those processes need to be notified so that they can decide how to proceed. The natural way of being notified is that the same exception is raised by **get**. A process doing an **await** is just interested in waiting for the computation to terminate, but not in knowing its result. Thus we claim that if the computation terminated, either with a normal value or with an exception, the **await** should not block and the exception, if any, should not be raised. The exception would be raised only if later on a **get** on the future is performed. This approach requires to put the fault notification inside the future, and has been explored in the context of ABS in [18]. Indeed, this is also the approach of Java future library (asynchronous computation with futures has been standardized in a Java library since Java SE 5 [11]). In contrast to ABS, Java's API does not distinguish between waiting for a future to become available, and retrieving the results. In fact, no primitive like **await** is available in Java. In addition, Java's futures do not faithfully propagate exceptions: the **get** method on a faulty future always raises the same exception `ExecutionException`.

An additional problem is related to concurrency. Indeed, in ABS, one may have multiple concurrent **get** and/or **await** statements on the same future containing an exception. Let us consider the case of multiple **get** statements. In this case, one has to decide whether they all raise the fault contained in the future or just one of them does. This second solution is more troublesome since to this end, the first process accessing the future would receive the exception and remove the fault from the future. The only possibility is to replace it with some default value, and this requires locking the future. However, this in turn changes the behavior of futures in a relevant way: Futures are understood as logical variables that change at most once, and this would no longer be true. Additionally, this creates a weak synchronization point between two processes accessing the same future. Indeed, if a process knows that a future originally contains an exception, by accessing it he will know if another process accessed it before. These weak synchronization points between processes that would be independent otherwise make the concurrency model and thus the analysis more difficult. Note that concurrent **await** statements are not a problem, since they do not locally raise the fault, but just check whether the future is empty or not.

Propagation through Method Invocations. It may be the case that the failed computation has invoked methods in other objects, whose execution is no more necessary after the failure. Indeed, it may even be undesired. For instance, if you are planning a trip and the booking of the airplane fails, you do not want to complete the booking of the hotel. Thus a mechanism to cancel a computation originated by a past method call may be useful. Actually, cancel may have different meanings according to the state of the invocation. If the invocation has not started yet, one can simply remove the invocation message itself. If the invoked process is running, one may raise the exception. If the execution already completed, one may do nothing or execute some code to compensate the terminated execution. This second option has been explored in [18]. The most interesting case is the one where the invoked process is running. Indeed, in this case the fault may be raised in any point of the execution, thus dealing with it using a try-catch would require to have the whole method code, including the return command, inside the try block. A better approach is to define specific points in the code where the running process checks for exceptions from its invoker, and specifying there the code to be executed in this case. A more modular way is to separate the two issues. One may have a statement `check` to specify when to check for faults, and a statement **setHandler** `H` establishing that `H` is the handler to be used to deal with faults from the invoker from now on. `H` can be a simple piece of code, or a function associating pieces of code to exceptions. Pieces of code may have a return statement, to communicate the result of the fault management to the invoker. If the execution of the handler terminates successfully, the execution of the method code restarts. One may also decide that the last handler has to be used to compensate the execution if the cancellation occurs after the termination of the invoked process.

We have described the effect of propagation to invoked processes. However, one has to understand which invoked processes to consider. The simplest possibility is to let the programmer decide. We call this approach *programmed propagation*. This can be done through a statement `f1 = f.`**cancel**`(e)` where `f` is the future corresponding to the invocation to be canceled, `e` the exception to be raised and `f1` the future storing the result of exception management. Note that the future `f` is the right entity to individuate the invocation, since each invocation corresponds to a different future. Note also that with programmed cancel one may cancel twice the same invocation, and that cancel can be executed by any process on any future he knows of. Future `f1` may contain different values according to the outcome of the cancel. If the exception sent by the cancel is correctly managed, the handler returns a specific value to fill that future (potentially different from the value returned as a result of the method, which is in future `f`). In all the other cases a system-defined exception is put inside future `f1` (one cannot put there a normal value, since this should depend on the type of the future):

– an exception `notStarted` if the **cancel** arrives before the invocation started (while the future `f` contains an exception `canceled`);
– an exception `terminated` if the **cancel** arrives after method termination, and compensations are not used (future `f` keeps its value);
– an exception `noCompensation` if the **cancel** arrives after method termination, compensations are allowed, but no compensation is specified for the target method (future `f` keeps its value);
– an exception `CancelNotManaged` if the exception arrives when the method is running, but it is not managed since there is no handler for it (while the future `f` stores the exception `e`).

In case of multiple cancellations, cancellations behave as above according to the state of the method when they are processed. Note also that the future `f` is not changed if it already contains the result of the method invocation.

An alternative approach is to have an *automatic propagation* of exceptions to invoked processes. First, one should decide whether to propagate only uncaught faults, or also managed faults. This last solution is not desirable in general, since most managed faults should not affect other processes, and can be dealt with by programmed propagation in case of need. Propagation of uncaught faults, if desired, should be necessarily done automatically. Now, the problem is to understand to which method invocations the fault needs to propagate. An upper bound is given by the futures known by the dying process. One may also consider that futures on which a **get** has already been performed are no more relevant. However, there is no fast and easy answer to this question. We think that a reasonable solution is to choose the futures which have been created by the current method execution and on which no **get** has been performed yet by the same method. One may also want to check whether the reference to the future has been passed to another method, and whether this method has performed a **get** on the future, but this would make the implementation and the analysis much more tricky. Similarly, one may want to consider also futures received as parameters, but again this needs to propagate the information on whether a future has been accessed or not from one method to the other. Note that in case of automatic propagation, no information on the result of the cancellation is needed, since the caller already terminated.

One may want to ensure that children can manage all the faults from their parent. To this end, each child should declare the exceptions that he can manage (at any point, since it may be the case that some exceptions can be managed only at some `check` due to handler modifications). Then, one can check that these include all the exceptions the parent may send to him. For automatic propagation, these coincide with the exceptions the parent may raise. For programmed propagation, these are the arguments of the various **cancel** of the corresponding future in the parent or in methods to which the future is passed.

Propagation to Other Processes in the same Object/Cog. We already discussed the fact that it is important for processes to restore the invariant of the object or cog before releasing the lock, and in particular before terminating because of an uncaught exception. In case the invariant cannot be restored, we proposed as a solution the possibility of killing the whole object/cog. An alternative solution is to terminate only the process P that first raised the fault, and notifying the other processes about the uncaught fault, since they may be able to manage it. Note that when process P is terminated, there is no other running process in the same cog. Thus the other processes will get the fault notification when they will be scheduled again. This means that they may get a fault, either when they start, or when they resume execution at an **await** or **suspend** statement. The fault may then be managed, or propagated as discussed above. In particular, if not managed, will be propagated to the next process to be scheduled. In this setting objects never die, but method calls may receive an exception as soon as they start. If we raise the same exception that was raised by P, then it may be difficult to track which exceptions may be raised inside any method. A simpler solution is to have a dedicated exception, e.g., InvariantNotRestored. What said above for objects holds similarly for cogs, concerning cog invariants. However, as already said, they are not a main concern in ABS at the moment.

Propagation through Dead Objects/Cogs. Some of the approaches we discussed involve the killing of an object or cog. We have not yet discussed what happens when a dead object is accessed (through method invocation). An exception should be raised. We can follow either the approach discussed above for network errors, or the one for normal faults. In practice the only difference is whether also the **await** will raise such a fault or not. We do not see any particular advantages or disadvantages for the two approaches: which is the best solution depends on which one better fits the programmer intuition, which may be different from one programmer to the other. In both the cases, using a standard exception such as DeadObject, instead of propagating the exception that caused the death of the object, simplifies the management. Also, it allows the caller to know whether the object is dead because of its invocation or it was already dead before.

Discussion. Among the propagation strategies above, propagation through the return future is nowadays standard, since it is used, e.g., in Java and C#. The possibility of canceling a running process via a future is also available in Java and C#, but the possibility of doing it automatically and/or of defining handlers and compensations for managing cancel requests while the process is running are not considered. Indeed, these strategies are quite complex, and it is not yet clear how useful they are in real programming. Also propagation of the fault to other processes in the same object is not considered in mainstream languages as far as we know, but we think this is a viable strategy in ABS. In fact, when a process is not able to restore the object invariant, there are two possibilities: either destroying the whole object or leave to another method call the task of restoring it. This second strategy seems also less extreme.

5 Conclusion

We have discussed the design space for fault models to be included in the ABS modeling language. As future work, we will extend the formal semantics of ABS with appropriate datatypes for the representation of faults, primitives to raise and catch faults, and mechanisms to distribute faults to other objects and cogs.

We complete the paper with a review of related fault models (the comparison with Java has been already discussed throughout the paper).

Functional programming languages, like OCaml or Haskell, also include primitives for faults modeled as exceptions. Both languages allow user-defined exceptions, but they implement them in different manners. OCaml uses a special type (called *exn*) to type exceptions, and new exceptions can be declared using the syntax **exception** *e* **of** *data*. In [24], the introduction of open datatypes in Haskell to encode exceptions is discussed. However, the current implementation of GHC uses *typeclass* [6], which allow one to register new datatypes as being exceptions.

Message-Passing Interface (MPI) is a cross-language standard, used, e.g., for C and Fortran, to program distributed applications. MPI expresses communication via so-called MPI functions, the basic ones being SEND and a blocking RECV. The SEND can work with three modalities: (*buffered send*) buffering the data to be sent, thus returning immediately as we assumed in this paper; (*synchronous send*) waiting for a corresponding RECV to be posted by the destination before terminating; or (*ready send*) failing in the case a corresponding RECV has not yet been posted by the destination. Dealing with the network, MPI functions represent communication at a lower level than we do: in MPI also the process of data delivering is taken into account. SEND (in all its modalities) and RECV have asynchronous variants, called ISEND and IRECV (where the "I" stands for "initiation"), which indicate a buffer where to fetch/put data and return immediately. For each such asynchronous send/receive, functionalities similar to some of the ones considered in this paper can be used: WAIT makes it possible to wait for the completion of data sending/receiving (in addition in MPI there is also a function TEST which returns immediately the status of the data sending/receiving without waiting); failures (e.g., in the communication while sending/receiving data) are detected by calling WAIT or TEST; and it is possible to cancel a send/receive by a call to CANCEL. The semantics of the latter, however, is the removal of the send/receive, supposing that it has not completed yet, as if it never occurred (a matching receive/send would perceive, as well, the canceled send/receive as if it never occurred). By combining the mechanisms above it is possible to obtain the waiting and canceling mechanisms considered in this paper. For example an asynchronous method invocation can be modeled by executing both ISEND and IRECV, and cancellation by executing CANCEL both on the send and the receive in the case the data is still under transmission or just on the receive in the case the send has completed. In the latter case, if the invoked method performs a ready send at the end of method execution, it will be notified of the matching IRECV having been canceled.

In web applications the HTTP protocol is used to realize service invocations by means of request/response pairs over a TCP/IP connection, as happens in the popular approaches of Java and Javascript, i.e. Asynchronous Javascript and XML (AJAX) invocations. In Java a method is used to initiate the HTTP request/response, which differently from the approach considered in this paper, may yield an error in the case the connection with the HTTP server cannot be established (timeout based). Then the client goes through a two phase process, where he first sends data over an output-stream and then, similarly, receives data. At the server side a symmetric process is followed. Java methods for reading pieces of response data are blocking as for the waiting function used in MPI and considered in this paper. Similarly, failures are notified via exceptions when reading response data (or while sending request data). Finally concerning cancellation, the HTTP request/response can be aborted as a whole by the client and this causes the server to detect the failure (an exception is raised) when it is in the phase of inserting data in the response, i.e. when returning data (or while reading request data). In Javascript (AJAX) request data are preliminarily put into memory (as in MPI) and then the request/response is initiated (again this can fail if connection cannot be established). Such an initiation function also installs a user-defined function which is expected to manage the data received once the response is completed (including also managing the case of failure). This mechanism is an alternative to the waiting function used in MPI and considered in this paper. Concerning cancellation, it is possible, as in Java, to abort the HTTP request/response (with the same effect at server side).

References

1. Agha, G., Hewitt, C.: Actors: A conceptual foundation for concurrent object-oriented programming. In: Research Directions in Object-Oriented Programming, pp. 49–74. MIT Press (1987)
2. Armstrong, J.: Programming Erlang: Software for a Concurrent World. Pragmatic Bookshelf (2007)
3. Baduel, L., et al.: Programming, Composing, Deploying, for the Grid. In: Grid Computing: Software Environments and Tools. Springer (2006)
4. Beckert, B., Hähnle, R., Schmitt, P.H.: Verification of Object-oriented Software: The KeY Approach. Springer (2007)
5. Caromel, D.: Service, Asynchrony, and Wait-By-Necessity. Journal of Object Oriented Programming, 12–22 (1989)
6. Chen, K., Hudak, P., Odersky, M.: Parametric type classes. In: Proc. of LFP 1992, pp. 170–181. ACM (1992)
7. Cristian, F.: Understanding fault-tolerant distributed systems. Communications of the ACM 34(2), 56–78 (1991)
8. de Boer, F.S., Clarke, D., Johnsen, E.B.: A complete guide to the future. In: De Nicola, R. (ed.) ESOP 2007. LNCS, vol. 4421, pp. 316–330. Springer, Heidelberg (2007)
9. Din, C.C., Dovland, J., Johnsen, E.B., Owe, O.: Observable behavior of distributed systems: Component reasoning for concurrent objects. Journal of Logic and Algebraic Programming 81(3), 227–256 (2012)
10. Elnozahy, E.N., Alvisi, L., Wang, Y.-M., Johnson, D.B.: A survey of rollback-recovery protocols in message-passing systems. ACM Computing Surveys 34(3), 375–408 (2002)
11. Goetz, B., Peierls, T., Bloch, J., Bowbeer, J., Holmes, D., Lea, D.: Java Concurrency in Practice. Addison-Wesley (2006)

12. Göri, G., Johnsen, E.B., Schlatte, R., Stolz, V.: Erlang-style error recovery for concurrent objects with cooperative scheduling. In: Margaria, T., Steffen, B. (eds.) ISoLA 2014, Part II. LNCS, vol. 8803, pp. 5–21. Springer, Heidelberg (2014)
13. Haller, P., Odersky, M.: Scala actors: Unifying thread-based and event-based programming. Theoretical Computer Science 410(2-3), 202–220 (2009)
14. Halstead Jr., R.H.: Multilisp: A language for concurrent symbolic computation. ACM Trans. Prog. Lang. Syst. 7(4), 501–538 (1985)
15. International Telecommunication Union. Open Distributed Processing — Reference Model parts 1–4. Technical report, ISO/IEC, Geneva (July 1995)
16. Jay, B.: Algebraic data types. In: Pattern Calculus, pp. 149–160. Springer (2009)
17. Johnsen, E.B., Hähnle, R., Schäfer, J., Schlatte, R., Steffen, M.: ABS: A core language for abstract behavioral specification. In: Aichernig, B.K., de Boer, F.S., Bonsangue, M.M. (eds.) FMCO 2010. LNCS, vol. 6957, pp. 142–164. Springer, Heidelberg (2011)
18. Johnsen, E.B., Lanese, I., Zavattaro, G.: Fault in the future. In: De Meuter, W., Roman, G.-C. (eds.) COORDINATION 2011. LNCS, vol. 6721, pp. 1–15. Springer, Heidelberg (2011)
19. JSR166: Concurrency utilities, `http://java.sun.com/j2se/1.5.0/docs/guide/concurrency`
20. Kramer, J.: Is abstraction the key to computing? Communications of the ACM 50(4), 36–42 (2007)
21. Lanese, I., Mezzina, C.A., Schmitt, A., Stefani, J.-B.: Controlling reversibility in higher-order pi. In: Katoen, J.-P., König, B. (eds.) CONCUR 2011. LNCS, vol. 6901, pp. 297–311. Springer, Heidelberg (2011)
22. Lavender, R.G., Schmidt, D.C.: Active object: an object behavioral pattern for concurrent programming. In: Pattern Languages of Program Design 2, pp. 483–499. Addison-Wesley Longman Publishing Co., Inc. (1996)
23. Liskov, B.H., Shrira, L.: Promises: Linguistic support for efficient asynchronous procedure calls in distributed systems. In: PLDI, pp. 260–267. ACM Press (1988)
24. Löh, A., Hinze, R.: Open data types and open functions. In: Proc. of PPDP 2006, pp. 133–144. ACM (2006)
25. Morris, B.: CActive and Friends. Symbian Developer Network (November 2007), `http://developer.symbian.com/main/downloads/papers/CActiveAndFriends/CActiveAndFriends.pdf`
26. Nienaltowski, P.: Practical framework for contract-based concurrent object-oriented programming. PhD thesis, Department of Computer Science, ETH Zurich (2007)
27. Pitt, E., McNiff, K.: Java.Rmi: The Remote Method Invocation Guide. Addison-Wesley Longman Publishing Co., Inc. (2001)
28. Ponge, J.: Better resource management with Java SE 7: Beyond syntactic sugar (May 2011), `http://www.oracle.com/technetwork/articles/java/trywithresources-401775.html`
29. Rémy, D.: Type checking records and variants in a natural extension of ml. In: Proc. of POPL 1989, pp. 77–88. ACM (1989)
30. Schäfer, J., Poetzsch-Heffter, A.: JCoBox: Generalizing active objects to concurrent components. In: D'Hondt, T. (ed.) ECOOP 2010. LNCS, vol. 6183, pp. 275–299. Springer, Heidelberg (2010)
31. Waldo, J., Wyant, G., Wollrath, A., Kendall, S.: A note on distributed computing. In: Vitek, J., Tschudin, C. (eds.) MOS 1996. LNCS, vol. 1222, pp. 49–64. Springer, Heidelberg (1997)
32. Yonezawa, A.: ABCL: An Object-Oriented Concurrent System. MIT Press (1990)

Programming with Actors in Java 8[⋆]

Behrooz Nobakht[1,2] and Frank S. de Boer[3]

[1] Leiden Advanced Institute of Computer Science
Leiden University
bnobakht@liacs.nl
[2] SDL Fredhopper
bnobakht@sdl.com
[3] Centrum Wiskunde en Informatica
frb@cwi.nl

Abstract. There exist numerous languages and frameworks that support an implementation of a variety of actor-based programming models in Java using concurrency utilities and threads. Java 8 is released with fundamental new features: lambda expressions and further dynamic invocation support. We show in this paper that such features in Java 8 allow for a high-level actor-based methodology for programming distributed systems which supports the programming to interfaces discipline. The embedding of our actor-based Java API is shallow in the sense that it abstracts from the actual thread-based deployment models. We further discuss different concurrent execution and thread-based deployment models and an extension of the API for its actual parallel and distributed implementation. We present briefly the results of a set of experiments which provide evidence of the potential impact of lambda expressions in Java 8 regarding the adoption of the actor concurrency model in large-scale distributed applications.

Keywords: Actor model, Concurrency, Asynchronous Message, Java, Lambda Expression.

1 Introduction

Java is beyond doubt one of the mainstream object oriented programming languages that supports a *programming to interfaces* discipline [9,35]. Through the years, Java has evolved from a mere programming language to a huge platform to drive and envision standards for mission-critical business applications. Moreover, the Java language itself has evolved in these years to support its community with new language features and standards. One of the noticeable domains of focus in the past decade has been distribution and concurrency in research and application. This has led to valuable research results and numerous libraries and frameworks with an attempt to provide distribution and concurrency at the level of Java language. However, it is widely recognized that the thread-based model of concurrency in Java that is a well-known approach is not

[⋆] This paper is funded by the EU project FP7-610582 ENVISAGE: Engineering Virtualized Services, `http://www.envisage-project.eu`.

appropriate for realizing distributed systems because of its inherent synchronous communication model. On the other hand, the event-driven actor model of concurrency introduced by Hewitt [17] is a powerful concept for modeling distributed and concurrent systems [2,1]. Different extensions of actors are proposed in several domains and are claimed to be the most suitable model of computation for many applications [18]. Examples of these domains include designing embedded systems [25,24], wireless sensor networks [6], multi-core programming [22] and delivering cloud services through SaaS or PaaS [5]. This model of concurrent computation forms the basis of the programming languages Erlang [3] and Scala [16] that have recently gained in popularity, in part due to their support for scalable concurrency. Moreover, based on the Java language itself, there are numerous libraries that provide an implementation of an actor-based programming model.

The main problem addressed in this paper is that in general existing actor-based programming techniques are based on an explicit encoding of mechanisms at the application level for message passing and handling, and as such overwrite the general object-oriented approach of method look-ups that forms the basis of programming to interfaces and the design-by-contract discipline [26]. The entanglement of event-driven (or asynchronous messaging) and object-oriented method look-up makes actor-based programs developed using such techniques extremely difficult to reason about and formalize. This clearly hampers the promotion of actor-based programming in mainstream industry that heavily practices object-oriented software engineering.

The main result of this paper is a Java 8 API for programming distributed systems using asynchronous message passing and a corresponding actor programming methodology which abstracts invocation from execution (e.g. thread-based deployment) and fully supports programming to interfaces discipline. We discuss the API architecture, its properties, and different concurrent execution models for the actual implementation.

Our main approach consists of the explicit description of an actor in terms of its *interface*, the use of the recently introduced lambda expressions in Java 8 in the implementation of asynchronous message passing, and the formalization of a corresponding high-level actor programming methodology in terms of an executable modeling language which lends itself to formal analysis, ABS [20].

The paper continues as follows: in Section 2, we briefly discuss a set of related works on actors and concurrent models especially on JVM platform. Section 3 presents an example that we use throughout the paper, we start to model the example using a library. Section 4 briefly introduces a concurrent modeling language and implements the example. Section 5 briefly discusses Java 8 features that this works uses for implementation. Section 6 presents how an actor model maps into programming in Java 8. Section 7 discusses in detail the implementation architecture of the actor API. Section 8 discusses how a number of benchmarks were performed for the implementation of the API and how they compare with current related works. Section 9 concludes the paper and discusses the future work.

2 Related Work

There are numerous works of research and development in the domain of actor modeling and implementation in different languages. We discuss a subset of the related work in the level of modeling and implementation with more focus on Java and JVM-based efforts in this section.

Erlang [3] is a programming language used to build massively scalable soft real-time systems with requirements on high availability. Some of its uses are in telecoms, banking, e-commerce, computer telephony and instant messaging. Erlang's runtime system has built-in support for concurrency, distribution and fault tolerance. While threads require external library support in most languages, Erlang provides language-level features for creating and managing processes with the aim of simplifying concurrent programming. Though all concurrency is explicit in Erlang, processes communicate using message passing instead of shared variables, which removes the need for locks. Elixir [33] is a functional meta-programming aware language built on top of the Erlang VM. It is a dynamic language with flexible syntax with macros support that leverages Erlang's abilities to build concurrent, distributed, fault-tolerant applications with hot code upgrades.

Scala is a hybrid object-oriented and functional programming language inspired by Java. The most important concept introduced in [16] is that Scala actors unify *thread-based* and *event-based* programming model to fill the gap for concurrency programming. Through the event-based model, Scala also provides the notion of continuations. Scala provides quite the same features of scheduling of tasks as in concurrent Java; i.e., it does not provide a direct and customizable platform to manage and schedule priorities on messages sent to other actors. Akka [15] is a toolkit and runtime for building highly concurrent, distributed, and fault tolerant event-driven applications on the JVM based on actor model.

Kilim [31] is a message-passing framework for Java that provides ultra-lightweight threads and facilities for fast, safe, zero-copy messaging between these threads. It consists of a bytecode postprocessor (a "weaver"), a run time library with buffered mailboxes (multi-producer, single consumer queues) and a user-level scheduler and a type system that puts certain constraints on pointer aliasing within messages to ensure interference-freedom between threads. The SALSA [34,22] programming language (Simple Actor Language System and Architecture) is an active object-oriented programming language that uses concurrency primitives beyond asynchronous message passing, including token-passing, join, and first-class continuations.

RxJava [7] by Netflix is an implementation of reactive extensions [27] from Microsoft. Reactive extensions try to provide a solution for composing asynchronous and event-based software using observable pattern and scheduling. An interesting direction of this library is that it uses reactive programming to avoid a phenomenon known as "callback hell"; a situation that is a natural consequence of composing `Future` abstractions in Java specifically when they wait for one another. However, RxJava advocates the use of asynchronous functions that are triggered in response to the other functions. In the same direction, LMAX Disruptor [4,8] is a highly concurrent event processing framework that takes the approach of event-driven programming towards provision of concurrency and asynchronous event handling. The system is built on the JVM platform and centers on a Business Logic Processor that can handle 6 million events per second on a single thread. The Business Logic Processor runs entirely in-memory using event sourcing. The Business Logic Processor is surrounded by Disruptors - a concurrency component that implements a network of queues that operate without needing locks.

3 State of the Art: An Example

In the following, we illustrate the state of the art in actor programming by means of a simple example using the Akka [32] library which features asynchronous messaging and which is used to program actors in both Scala and Java. We want to model in Akka an "asynchronous ping-pong match" between two actors represented by the two interfaces `IPing` and `IPong` which are depicted in Listings 1 and 2. An asynchronous call by the actor implementing the `IPong` interface of the `ping` method of the actor implementing the `IPing` interface should generate an asynchronous call of the `pong` method of the callee, and vice versa. We intentionally design `ping` and `pong` methods to take arguments in order to demonstrate how method arguments may affect the use of an actor model in an object-oriented style.

Listing 1. Ping as an interface

```java
public interface IPing {
  void ping(String msg);
}
```

Listing 2. Pong as an interface

```java
public interface IPong {
  void pong(String msg);
}
```

To model an actor in Akka by a class, say `Ping`, with interface `IPing`, this class is required *both* to *extend* a given pre-defined class `UntypedActor` and *implement* the interface `IPing`, as depicted in Listings 3 and 4. The class `UntypedActor` provides two Akka framework methods `tell` and `onReceive` which are used to enqueue and dequeue asynchronous messages. An asynchronous call to, for example, the method `ping` then can be modeled by passing a user-defined encoding of this call, in this case by prefixing the string argument with the string "pinged", to a (synchronous) call of the `tell` method which results in enqueuing the message. In case this message is dequeued the implementation of the `onReceive` method as provided by the `Ping` class then calls the `ping` method.

Listing 3. Ping actor in Akka

```java
public class Ping(ActorRef pong)
  extends UntypedActor
  implements IPing {

  public void ping(String msg) {
    pong.tell("ponged," + msg)
  }

  public void onReceive(Object m)
       {
    if (!(m instanceof String)) {
      // Message not understood.
    } else
    if (((String) m).startsWith("
        pinged") {
      // Explicit cast needed.
      ping((String) m);
    }
   }
}
```

Listing 4. Pong class in Akka

```java
public class Pong
  extends UntypedActor
  implements IPong {

  public void pong(String msg) {
    sender().tell(
      "pinged," + msg);
  }

  public void onReceive(Object m)
       {
    if (!(m instanceof String)) {
      // Message not understood.
    } else
    if (m.startsWith("ponged") {
      // Explicit cast needed.
      ping((String) m);
    }
   }
}
```

Access to the sender of the message in Akka is provided by `sender()`. In the main method as described in Listing 5 we show how the initialize and start the ping/pong match. Note that a reference to the "pong" actor is passed to the "ping" actor.

Listing 5. main in Akka

```java
ActorSystem s = ActorSystem.create
     ();
ActorRef pong = s.actorOf(Props.
     create(Pong.class));
ActorRef ping = s.actorOf(Props.
     create(Ping.class, pong));
ping.tell(""); // To get a Future
```

Further, both the `onReceive` methods are invoked by Akka `ActorSystem` itself. In general, Akka actors are of type `ActorRef` which is an abstraction provided by Akka to allow actors send asynchronous messages to one another. An immediate consequence of the above use of inheritance is that the class `Ping` is now exposing a public behavior that is *not* specified by its *interface*. Furthermore, a "ping" object refers to a "pong" object by the type `ActorRef` . This means that the interface `IPong` is not directly visible to the "ping" actor. Additionally, the implementation details of receiving a message should be "hand coded" by the programmer into the special method `onReceive` to define the responses to the received messages. In our case, this implementation consists of a decoding of the message (using type-checking) in order to *look up* the method that subsequently should be invoked. This fundamentally interferes with the general object-oriented mechanism for method look-up which forms the basis of the programming to interfaces discipline. In the next section, we continue the same example and discuss an actor API for directly calling asynchronously methods using the general object-oriented mechanism for method look-up. Akka has recently released a new version that supports Java 8 features [1]. However, the new features can be categorized as syntax sugar on how incoming messages are filtered through object/class matchers to find the proper type.

[1] Documentation available at `http://doc.akka.io/docs/akka/2.3.2/java/lambda-index-actors.html`

4 Actor Programming in Java

We first describe informally the actor programming model assumed in this paper. This model is based on the Abstract Behavioral Specification language (ABS) introduced in [20]. ABS uses asynchronous method calls, futures, interfaces for encapsulation, and cooperative scheduling of method invocations inside concurrent (active) objects. This feature combination results in a concurrent object-oriented model which is inherently compositional. More specifically, actors in ABS have an identity and behave as active objects with encapsulated data and methods which represent their state and behavior, respectively. Actors are the units of concurrency: conceptually an actor has a dedicated processor. Actors can only send asynchronous messages and have queues for receiving messages. An actor progresses by taking a message out of its queue and processing it by executing its corresponding method. A method is a piece of sequential code that may send messages. Asynchronous method calls use futures as dynamically generated references to return values. The execution of a method can be (temporarily) suspended by release statements which give rise to a form of cooperative scheduling of method invocations inside concurrent (active) objects. Release statements can be conditional (e.g., checking a future for the return value). Listings 7, 8 and 6 present an implementation of ping-pong example in ABS. By means of the statement on line 6 of Listing 7 a "ping" object directly calls asynchronously the `pong` method of its "pong" object, and vice versa. Such a call is stored in the message queue and the called method is executed when the message is dequeued. Note that variables in ABS are declared by interfaces. In ABS, `Unit` is similar to `void` in Java.

Listing 6. main in ABS

```
ABSIPong pong;
pong = new ABSPong;
ping = new ABSPing(pong);
ping ! ping("");
```

Listing 7. Ping in ABS

```
1 interface ABSIPing {
2   Unit ping(String msg);
3 }
4 class ABSPing(ABSIPong pong)
       implements ABSIPing {
5   Unit ping(String msg) {
6   pong ! pong("ponged," + msg);
7   }
8 }
```

Listing 8. Pong in ABS

```
interface ABSIPong {
  Unit pong(String msg);
}
class ABSPong implements ABSIPong
     {
  Unit pong(String msg) {
    sender ! ping( "pinged," + msg
         );
  }
}
```

5 Java 8 Features

In the next section, we describe how ABS actors are implemented in Java 8 as API. In this section we provide an overview of the features in Java 8 that facilitate an efficient, expressive, and precise implementation of an actor model in ABS.

Java Defender Methods Java *defender* methods (JSR 335 [13]) use the new keyword **`default`**. Defender methods are declared for **`interface`**s in Java. In contrast to the other methods of an interface, a default method is not an abstract method but must have an implementation. From the perspective of a client of the interface, defender methods are no different from ordinary interface methods. From the perspective of a hierarchy descendant, an implementing class can optionally *override* a default method and change the behavior. It is left as a decision to any class implementing the interface whether or not to override the default implementation. For instance, in Java 8 `java.util.Comparator` provides a default method `reversed()` that creates a reversed-order comparator of the original one. Such default method eliminates the need for any implementing class to provide such behavior by inheritance.

Java Functional Interfaces Functional interfaces and lambda expressions (JSR 335 [13]) are fundamental changes in Java 8. A **`@FunctionalInterface`** is an annotation that can be used for interfaces in Java. Conceptually, any class or interface is a functional interface if it consists of exactly one *abstract* method. A lambda expression in Java 8, is a runtime translation [11] of any type that is replaceable by a functional interface. Many of Java's classic interfaces are functional interfaces from the perspective of Java 8 and can be turned into lambda expressions; e.g. `java.lang.Runnable` or `java.util.Comparator`. For instance,

```
(s1, s2) → return s1.compareTo(s2);
```

is a lambda expression that can be statically cast to an instance of a `Comparator<String>`; because it can be replaced with a functional interface that has a method with two strings and returning one integer. Lambda expressions in Java 8 *do not* have an intrinsic type. Their type is bound to the context that they are used in but their type is always a functional interface. For instance, the above definition of a lambda expression can be used as:

```
Comparator<String> cmp1 = (s1, s2) → return s1.compareTo(s2);
```

in one context while in the other:

```
Function<String> cmp2 = (s1, s2) → return s1.compareTo(s2);
```

given that `Function<T>` is defined as:

```
interface Function<T> { int apply(T t1, T t2); }
```

In the above examples, the same lambda expression is statically cast to a different matching functional interface based on the context. This is a fundamental new feature in Java 8 that facilitates application of functional programming paradigm in an object-oriented language.

This work of research extensively uses this feature of Java 8. Java 8 marks many of its own APIs as functional interfaces most important of which in this context are `java.lang.Runnable` and `java.util.concurrent.Callable`. This means that a lambda expression can replace an instance of `Runnable` or `Callable` at runtime by JVM. We will discuss later how we utilize this feature to allow us model an asynchronous message into an instance of a `Runnable` or `Callable` as a form of a lambda expression. A lambda expression equivalent of a `Runnable` or a `Callable` can be treated as a queued message of an actor and executed.

Java Dynamic Invocation Dynamic invocation and execution with method handles (JSR 292 [29]) enables JVM to support efficient and flexible execution of method invocations in the absence of static type information. JSR 292 introduces a new byte code instruction `invokedynamic` for JVM that is available as an API through `java.lang.invoke.MethodHandles`. This API allows translation of lambda expression in Java 8 at runtime to be executed by JVM. In Java 8, use of lambda expression are favored over anonymous inner classes mainly because of their performance issues [12]. The abstractions introduced in JSR 292 perform better that Java Reflection API using the new byte code instruction. Thus, lambda expressions are compiled and translated into method handle invocations rather reflective code or anonymous inner classes. This feature of Java 8 is indirectly use in ABS API through the extensive use of lambda expressions. Moreover, in terms of performance, it has been revealed that invoke dynamic is much better than using anonymous inner classes [12].

6 Modeling Actors in Java 8

In this section, we discuss how we model ABS actors using Java 8 features. In this mapping, we demonstrate how new features of Java 8 are used.

The `Actor` Interface. We introduce an interface to model actors using Java 8 features discussed in Section 5. Implementing an interface in Java means that the object exposes public APIs specified by the interface that is considered the behavior of the object. Interface implementation is opposed to inheritance extension in which the object is possibly forced to expose behavior that may not be part of its intended interface. Using an interface for an actor allows an object to preserve its own interfaces, and second, it allows for multiple interfaces to be implemented and composed.

A Java API for the implementation of ABS models should have the following main three features. First, an object should be able to send asynchronously an arbitrary message in terms of a method invocation to a receiver actor object. Second, sending a message can optionally generate a so-called future which is used to refer to the return value. Third, an object during the processing of a message should be able to access the “sender” of a message such that it can reply to the message by another message. All the above must co-exist with the fundamental requirement that for an object to act like an actor (in an object-oriented context) should *not* require a modification of its intended interface.

The `Actor` interface (Listings 9 and 10) provides a set of **`default`** methods, namely the `run` and `send` methods, which the implementing classes do not need to re-implement. This interface further encapsulates a queue of messages that supports concurrent features of Java API [2]. We distinguish two types of messages: messages that are not

[2] Such API includes usage of different interfaces and classes in `java.util.concurrent` package [23]. The concurrent Java API supports blocking and synchronization features in a high-level that is abstracted from the user.

expected to generate any result and messages that are expected to generate a result captured by a future value; i.e. an instance of `Future` in Java 8. The first kind of messages are modeled as instances of `Runnable` and the second kind are modeled instances of `Callable`. The default `run` method then takes a message from the queue, checks its type and executes the message correspondingly. On the other hand, the default (overloaded) `send` method stores the sent message and creates a future which is returned to the caller, in case of an instance of `Callable`.

Listing 9. Actor interface (1)

```
public interface Actor {
  public void run() {
    Object m = queue.take();

    if (m instanceof Runnable) {
      ((Runnable) m).run();
    } else

    if (m instanceof Callable) {
      ((Callable) m).call();
    }
  }

  // continue to the right
```

Listing 10. Actor interface (2)

```

  public void send(Runnable m) {
    queue.offer(m);
  }

  public <T> Future<T>
    send(Callable<T> m) {
      Future<T> f =
        new FutureTask(m);
      queue.offer(f);
      return f;
  }
}
```

Modeling Asynchronous Messages We model an asynchronous call

$$\texttt{Future<V>}\ f = e_0\ !\ m(e_1, \ldots, e_n)$$

to a method in ABS by the Java 8 code snippet of Listing 11. The final local variables $\texttt{u}_1, \ldots, \texttt{u}_n$ (of the caller) are used to store the values of the Java 8 expressions $\texttt{e}_1, \ldots, \texttt{e}_n$ corresponding to the actual parameters $e_1, \ldots, e_n$. The types $\texttt{T}_i$, $i = 1, \ldots, n$, are the corresponding Java 8 types of e_i, $i = 1, \ldots, n$.

Listing 11. Async messages with futures

```
final T₁ u₁ = e₁;
. . .
final Tₙ uₙ = eₙ;
Future<V> v = e₀.send(
  () → { return m(u₁,...,uₙ); }
);
```

Listing 12. Async messages w/o futures

```
final T₁ u₁ = e₁;
. . .
final Tₙ uₙ = eₙ;
e₀.send(
  { () → m(u₁,...,uₙ); }
);
```

The lambda expression which encloses the above method invocation is an instance of the functional interface; e.g. `Callable`. Note that the generated object which represents the lambda expression will contain the local context of the caller of the method "m" (including the local variables storing the values of the expressions $e_1, \ldots, e_n$), which will be restored upon execution of the lambda expression. Listing 12 models an asynchronous call to a method without a return value.

As an example, Listings 13 and 14 present the running ping/pong example, using the above API. The main program to use ping and pong implementation is presented in Listing 15.

Listing 13. Ping as an Actor

```
public class Ping(IPong pong)
     implements IPing, Actor {
 public void ping(String msg) {
  pong.send( () → { pong.("ponged
       ," + msg) } );
  }
}
```

Listing 14. Pong as an Actor

```
public class Pong implements IPong
     , Actor {
  public void pong(String msg) {
    sender().send(( ) → { ping.("
        pinged," + msg) } );
  }
}
```

As demonstrated in the above examples, the "ping" and "pong" objects *preserve* their own *interfaces* contrary to the example depicted in Section 3 in which the objects *extend* a specific "universal actor abstraction" to inherit methods and behaviors to become an actor. Further, messages are processed *generically* by the `run` method described in Listing 9. Although, in the first place, sending an asynchronous may look like to be able to change the recipient actor's state, this is not correct. The variables that can be used in a lambda expression are *effectively* final. In other words, in the context of a lambda expression, the recipient actor only provides a snapshot view of its state that cannot be changed. This prevents abuse of lambda expressions to change the receiver's state.

Modeling Cooperative Scheduling The ABS statement **await** g, where g is a boolean guard, allows an active object to preempt the current method and schedule another one. We model cooperative scheduling by means of a call to the `await` method in Listing 16. Note that the preempted process is thus passed as an additional parameter and as such queued in case the guard is false, otherwise it is executed. Moreover, the generation of the continuation of the process is an optimization task for the code generation process to prevent code duplication.

Listing 15. main in ABS API

```
IPong pong = new Pong();
IPing ping = new Ping(pong);
ping.send(
  () -> ping.ping("")
);
```

Listing 16. Java 8 await implementation

```
void await(final Boolean guard,
           final Runnable cont) {
  if (!guard) {
    this.send(() →
      { this.await(guard, cont) })
  } else { cont.run() }
}
```

7 Implementation Architecture

Figure 1 presents the general layered architecture of the actor API in Java 8. It consists of three layers: the routing layer which forms the foundation for the support of distribution and location transparency [22] of actors, the queuing layer which allows for different implementations of the message queues, and finally, the processing layer which implements the actual execution of the messages. Each layer allows for further customization by means of plugins. The implementation is available at `https://github.com/CrispOSS/abs-api`.

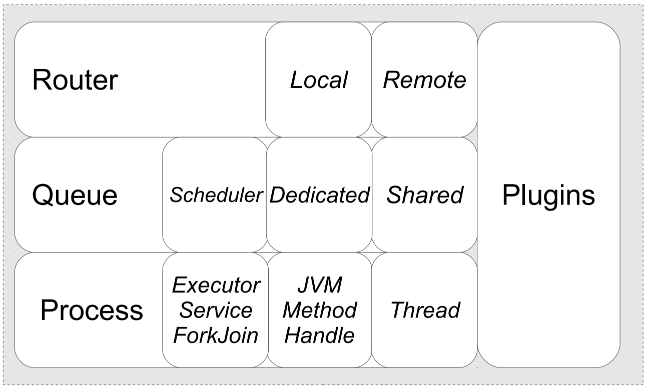


Fig. 1. Architecture of Actor API in Java 8

We discuss the architecture from bottom layer to top. The implementation of actor API preserves a faithful mapping of message processing in ABS modeling language. An actor is an active object in the sense that it controls how the next message is executed and may release any resources to allow for co-operative scheduling. Thus, the implementation is required to optimally utilize JVM threads. Clearly, allocating a dedicated thread to each message or actor is not scalable. Therefore, actors need to *share* threads for message execution and yet be in full control of resources when required. The implementation fundamentally separates *invocation* from *execution*. An asynchronous message is a reference to a method invocation until it starts its execution. This allows to minimize the allocation of threads to the messages and facilitates sharing threads for executing messages. Java concurrent API [23] provides different ways to deploy this separation of invocation from execution. We take advantage of Java Method Handles [29] to encapsulate invocations. Further we utilize different forms of `ExecutorService` and `ForkJoinPool` to deploy concurrent invocations of messages in different actors.

In the next layer, the actor API allows for different implementations of a queue for an actor. A dedicated queue for each actor simplifies the process of queuing messages for execution but consumes more resources. However, a shared queue for a set of actors allows for memory and storage optimization. This latter approach of deployment, first, provides a way to utilize the computing power of multi-core; for instance, it allows to use work-stealing to maximize the usage of thread pools. Second, it enables application-level scheduling of messages. The different implementations cater for a variety of plugins, like one that releases computation as long as there is no item in the queue and becomes active as soon as an item is placed into the queue; e.g.

`java.util.concurrent.BlockingQueue`. Further, different plugins can be injected to allow for scheduling of messages extended with deadlines and priorities [28].

We discuss next the distribution of actors in this architecture. In the architecture presented in Figure 1, each layer can be *distributed* independently of another layer in a transparent way. Not only the routing layer can provide distribution, the queue layer of the architecture may also be remote to take advantage of cluster storage for actor messages. A remote routing layer can provide access to actors transparently through standard naming or addresses. We exploit the main properties of actor model [1,2] to distribute actors based on our implementation. From a distributed perspective, the following are the main requirements for distributing actors:

Reference Location Transparency. Actors communicate to one another using references. In an actor model, there is no in-memory object reference; however, every actor reference denotes a location by means of which the actor is accessible. The reference location may be local to the calling actor or remote. The reference location is *physically* transparent for the calling actor.

Communication Transparency. A message m from actor A to actor B may possibly lead to transferring m over a network such that B can process the message. Thus, an actor model that supports distribution must provide a layer of remote communication among its actors that is transparent, i.e., when actor A sends message m, the message is transparently transferred over the network to reach actor B. For instance, actors existing in an HTTP container that transparently allows such communication. Further, the API implementation is required to provide a mechanism for serialization of messages. By default, every object in JVM cannot be assumed to be an instance of `java.io.Serializable`. However, the API may enforce that any remote actor should have the required actor classes in its JVM during runtime which allows the use of the JVM's general object serialization [3] to send messages to remote actors and receive their responses. Additionally, we model asynchronous messages with lambda expressions for which Java 8 supports serialization by specification [4].

Actor Provisioning. During a life time of an actor, it may need to create new actors. Creating actors in a local memory setting is straightforward. However, the local setting *does* have a capacity of number of actors it can hold. When an actor creates a new one, the new actor may actually be initialized in a remote resource. When the resource is not available, it should be first provisioned. However, this resource provisioning should be transparent to the actor and only the eventual result (the newly created actor) is visible.

We extend the ABS API to ABS Remote API[5] that provides the above properties for actors in a seamless way. A complete example of using the remote API has been

[3] Java Object Serialization Specification: `http://docs.oracle.com/javase/8/docs/platform/serialization/spec/serialTOC.html`

[4] Serialized Lambdas: `http://docs.oracle.com/javase/8/docs/api/java/lang/invoke/SerializedLambda.html`

[5] The implementation is available at `https://github.com/CrispOSS/abs-api-remote`.

developed[6]. Expanding our ping-pong example in this paper, Listing 17 and 18 present how a remote server of actors is created for the ping and pong actors. In the following listings, `java.util.Properties` is used provide input parameters of the actor server; namely, the address and the port that the actor server responds to.

Listing 17. Remote ping actor main

```
Properties p =  new Properties();
p.put("host", "localhost");
p.put("port", "7777");
ActorServer s = new ActorServer(p)
     ;
IPong pong =
 s.newRemote("abs://pong@http://
      localhost:8888",
  IPong.class);
Ping ping = new Ping(pong);
ping.send(
  () -> ping.ping("")
);
```

Listing 18. Remote pong actor main

```
Properties p = new Properties();
p.put("host", "localhost");
p.put("port", "8888");
ActorServer s = new ActorServer(p)
     ;
Pong pong = new Pong();
```

In Listing 17, a remote reference to a pong actor is created that exposes the `IPong` interface. This interface is proxied [7] by the implementation to handle the remote communication with the actual pong actor in the other actor server. This mechanism hides the communication details from the ping actor and as such allows the ping actor to use the same API to send a message to the pong actor (without even knowing that the pong actor is actually remote). When an actor is initialized in a distributed setting it transparently identifies its actor server and registers with it. The above two listings are aligned with the similar main program presented in Listing 15 that presents the same in a local setting. The above two listings run in separate JVM instances and therefore do not share any objects. In each JVM instance, it is required that both interfaces `IPing` and `IPong` are visible to the classpath; however, the ping actor server only needs to see `Ping` class in its classpath and similarly the pong actor server only needs to see `Pong` class in its classpath.

8 Experiments

In this section, we explain how a series of benchmarks were directed to evaluate the performance and functionality of actor API in Java 8. For this benchmark, we use a simple Java application that uses the "Ping-Pong" actor example discussed previously. An application consists of one instance of `Ping` actor and one instance of `Pong` actor. The application sends a `ping` message to the ping actor and waits for the result. The ping message depends on a `pong` message to the pong actor. When the result from the pong

[6] An example of ABS Remote API is available at `https://github.com/CrispOSS/abs-api-remote-sample`.

[7] Java Proxy: `http://docs.oracle.com/javase/8/docs/api/java/lang/reflect/Proxy.html`

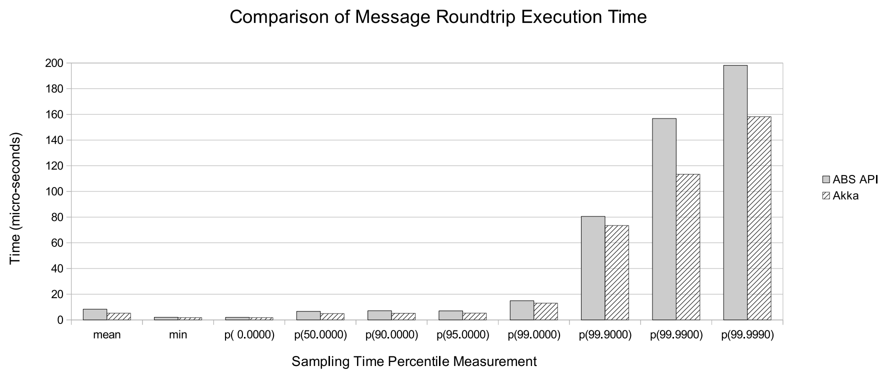


Fig. 2. Benchmark results of comparing sampling time of message round trips in ABS API and Akka. An example reading of above results is that the time shows for $p(90.0000)$ reads as "message round trips were completed under $10\mu s$ for 90% of the sent messages". The first two columns show the "minimum" and "mean" message round trip times in both implementations.

actor is ready, the ping actor completes the message; this completes a round trip of a message in the application. To be able to make comparison of how actor API in Java 8 performs, the example is also implemented using Akka [32] library. The same set of benchmarks are performed in isolation for both of the applications. To perform the benchmarks, we use JMH [30] that is a Java microbenchmarking harness developed by OpenJDK community and used to perform benchmarks for the Java language itself.

The benchmark is performed on the round trip of a message in the application. The benchmark starts with a warm-up phase followed by the running phase. The benchmark composes of a number of iterations in each phase and specific time period for each iteration specified for each phase. Every iteration of the benchmark triggers a new message in the application and waits for the result. The measurement used is *sampling time* of the round trip of a message. A specific number of samples are collected. Based on the samples in different phases, different *percentile* measurements are summarized. An example percentile measurement $p(99.9900) = 10\ \mu s$ is read as 99.9900% of messages in the benchmark took 10 micro-seconds to complete.

Each benchmark starts with 500 iterations of warm-up with each iteration for 1 micro-second. Each benchmark runs for 5000 iterations with each iteration for 50 micro-seconds. In each iteration, a maximum number of 50K samples are collected. Each benchmark is executed in an isolated JVM environment with Java 8 version b127. Each benchmark is executed on a hardware with 8 cores of CPU and a maximum memory of 8GB for JVM.

The results are presented in Figure 2. The performance difference observed in the measurements can be explained as follows. An actor in Akka is expected to expose a certain behavior as discussed in Section 3 (i.e. `onReceive`). This means that every message leads to an eventual invocation of this method inside actor. However, in case of an actor in Java 8, there is a need to make a look-up for the actual method to be executed with expected arguments. This means that for every method, although in the presence of caching, there is a need to find the proper method that is expected to be invoked. A constant overhead for the method look-up in order to adhere to the object-oriented principles is naturally to be expected. Thus, this is the minimal performance cost that the actor API in Java 8 pays to support programming to interfaces.

9 Conclusion

In this paper, we discussed an implementation of the actor-based ABS modeling language in Java 8 which supports the basic object-oriented mechanisms and principles of method look-up and programming to interfaces. In the full version of this paper we have developed an operational semantics of Java 8 features including lambda expressions and have proved formally the correctness of the embedding in terms of a bisimulation relation.

The underlying modeling language has an executable semantics and supports a variety of formal analysis techniques, including deadlock and schedulability analysis [10,19]. Further it supports a formal behavioral specification of interfaces [14], to be used as contracts.

We intend to expand this work in different ways. We aim to automatically *generate* ABS models from Java code which follows the ABS design methodology. Model extraction allows industry level applications be abstracted into models and analyzed for different goals such as deadlock analysis and concurrency optimization. This approach of model extraction we believe will greatly enhance industrial uptake of formal methods. We aim to further extend the implementation of API to support different features especially regarding distribution of actors especially in the queue layer, and scheduling of messages using application-level policies or real-time properties of a concurrent system. Furthermore, the current implementation of ABS API in a distributed setting allows for instantiation of remote actors. We intend to use the implementation to model ABS deployment components [21] and simulate a distributed environment.

References

1. Agha, G., Mason, I., Smith, S., Talcott, C.: A Foundation for Actor Computation. Journal of Functional Programming 7, 1–72 (1997)
2. Agha, G.: The Structure and Semantics of Actor Languages. In: de Bakker, J.W., Rozenberg, G., de Roever, W.-P. (eds.) REX 1990. LNCS, vol. 489, pp. 1–59. Springer, Heidelberg (1991)
3. Armstrong, J.: Erlang. Communications of ACM 53(9), 68–75 (2010)
4. Baker, M., Thompson, M.: LMAX Disruptor. LMAX Exchange, `http://github.com/LMAX-Exchange/disruptor`
5. Chang, P.-H., Agha, G.: Towards Context-Aware Web Applications. In: Indulska, J., Raymond, K. (eds.) DAIS 2007. LNCS, vol. 4531, pp. 239–252. Springer, Heidelberg (2007)
6. Cheong, E., Lee, E.A., Zhao, Y.: Viptos: a graphical development and simulation environment for tinyOS-based wireless sensor networks. In: Proc. Embedded Net. Sensor Sys., SenSys 2005, p. 302 (2005)
7. Christensen, B.: RxJava: Reactive Functional Progamming in Java. Netflix, `http://github.com/Netflix/RxJava/wiki`

8. Fowler, M.: LMAX Architecture. Martin Fowler, http://martinfowler.com/articles/lmax.html
9. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Abstraction and Reuse of Object-Oriented Design. In: Nierstrasz, O.M. (ed.) ECOOP 1993. LNCS, vol. 707, pp. 406–431. Springer, Heidelberg (1993)
10. Giachino, E., Grazia, C.A., Laneve, C., Lienhardt, M., Wong, P.Y.H.: Deadlock analysis of concurrent objects: Theory and practice. In: Johnsen, E.B., Petre, L. (eds.) IFM 2013. LNCS, vol. 7940, pp. 394–411. Springer, Heidelberg (2013)
11. Goetz, B.: Lambda Expression Translation in Java 8. Oracle, http://cr.openjdk.java.net/˜briangoetz/lambda/lambda-translation.html
12. Goetz, B.: Lambda: A Peek Under The Hood. Oracle, JAX London (2012)
13. Goetz, B.: JSR 335, Lambda Expressions for the Java Programming Language. Oracle (March 2013), http://jcp.org/en/jsr/detail?id=335
14. Hähnle, R., Helvensteijn, M., Johnsen, E.B., Lienhardt, M., Sangiorgi, D., Schaefer, I., Wong, P.Y.H.: HATS abstract behavioral specification: The architectural view. In: Beckert, B., Damiani, F., de Boer, F.S., Bonsangue, M.M. (eds.) FMCO 2011. LNCS, vol. 7542, pp. 109–132. Springer, Heidelberg (2013)
15. Haller, P.: On the integration of the actor model in mainstream technologies: the Scala perspective. In: Proceedings of the 2nd Edition on Programming Systems, Languages and Applications Based on Actors, Agents, and Decentralized Control Abstractions, pp. 1–6. ACM (2012)
16. Haller, P., Odersky, M.: Scala Actors: Unifying thread-based and event-based programming. Theoretical Computer Science 410(2-3), 202–220 (2009)
17. Hewitt, C.: Procedural Embedding of knowledge in Planner. In: Proc. the 2nd International Joint Conference on Artificial Intelligence, pp. 167–184 (1971)
18. Hewitt, C.: What Is Commitment? Physical, Organizational, and Social (Revised). In: Noriega, P., Vázquez-Salceda, J., Boella, G., Boissier, O., Dignum, V., Fornara, N., Matson, E. (eds.) COIN 2006 Workshops. LNCS (LNAI), vol. 4386, pp. 293–307. Springer, Heidelberg (2007)
19. Jaghoori, M.M., de Boer, F.S., Chothia, T., Sirjani, M.: Schedulability of asynchronous real-time concurrent objects. J. Log. Algebr. Program. 78(5), 402–416 (2009)
20. Johnsen, E.B., Hähnle, R., Schäfer, J., Schlatte, R., Steffen, M.: ABS: A core language for abstract behavioral specification. In: Aichernig, B.K., de Boer, F.S., Bonsangue, M.M. (eds.) FMCO 2010. LNCS, vol. 6957, pp. 142–164. Springer, Heidelberg (2011)
21. Johnsen, E.B., Schlatte, R., Tapia Tarifa, S.L.: Modeling resource-aware virtualized applications for the cloud in Real-Time ABS. In: Aoki, T., Taguchi, K. (eds.) ICFEM 2012. LNCS, vol. 7635, pp. 71–86. Springer, Heidelberg (2012)
22. Karmani, R.K., Shali, A., Agha, G.: Actor frameworks for the JVM platform: a comparative analysis. In: Proc. Principles and Practice of Prog. in Java (PPPJ 2009), pp. 11–20. ACM (2009)
23. Lea, D.: JSR 166: Concurrency Utilities. Sun Microsystems, Inc., http://jcp.org/en/jsr/detail?id=166
24. Lee, E.A., Liu, X., Neuendorffer, S.: Classes and inheritance in actor-oriented design. ACM Transactions in Embedded Computing Systems 8(4) (2009)
25. Lee, E.A., Neuendorffer, S., Wirthlin, M.J.: Actor-Oriented Design of Embedded Hardware and Software Systems. Journal of Circuits, Systems, and Computers 12(3), 231–260 (2003)
26. Meyer, B.: Applying "design by contract". Computer 25(10), 40–51 (1992)
27. Microsoft. Reactive Extensions. Microsoft, https://rx.codeplex.com/

28. Nobakht, B., de Boer, F.S., Jaghoori, M.M., Schlatte, R.: Programming and deployment of active objects with application-level scheduling. In: Proceedings of the 27th Annual ACM Symposium on Applied Computing, SAC 2012, pp. 1883–1888. ACM (2012)
29. Rose, J.: JSR 292: Supporting Dynamically Typed Languages on the Java Platform. Oracle, http://jcp.org/en/jsr/detail?id=292
30. Shipilev, A.: JMH: Java Microbenchmark Harness. Oracle, http://openjdk.java.net/projects/code-tools/jmh/
31. Srinivasan, S., Mycroft, A.: Kilim: Isolation-typed actors for Java. In: Vitek, J. (ed.) ECOOP 2008. LNCS, vol. 5142, pp. 104–128. Springer, Heidelberg (2008)
32. Typesafe. Akka. Typesafe, http://akka.io/
33. Valim, J.: Elixir. Elixir, http://elixir-lang.org/
34. Varela, C.A., Agha, G., Wang, W.-J., Desell, T., El Maghraoui, K., LaPorte, J., Stephens, A.: The SALSA Programming Language 1.1.2 Release Tutorial. Dept. of Computer Science, RPI, Tech. Rep., pp. 07–12 (2007)
35. Wirfs-Brock, R.J., Johnson, R.E.: Surveying Current Research in Object-oriented Design. Commun. ACM 33(9), 104–124 (1990)

Contracts in CML

Jim Woodcock[1], Ana Cavalcanti[1], John Fitzgerald[2], Simon Foster[1], and Peter Gorm Larsen[3]

[1] University of York
[2] Newcastle University
[3] Aarhus University

Abstract. We describe the COMPASS Modelling Language (CML), which is used to model large-scale Systems of Systems and the contracts that bind them together. The language can be used to document the interfaces to constituent systems using formal, precise, and verifiable specifications including preconditions, postconditions, and invariants. The semantics of CML directly supports the use of these contracts for all language constructs, including the use of communication channels, parallel processes, and processes that run forever. Every process construct in CML has an associated contract, allowing clients and suppliers to check that the implementations of constituent systems conform to their interface specifications.

1 Introduction

The COMPASS Modelling Language (CML) is a formal language for describing "Systems of Systems" [29] (SoS). An SoS is a collaboration of smaller independent systems to achieve a goal that cannot be readily achieved by any of these constituents. Typical SoSs include traffic management systems, emergency response systems, and home automation systems, all of which include constituent systems over which there is no overall control. To achieve synergy, contracts must be negotiated to define the behaviour of the SoS and impose constraints on the constituents, which are otherwise free to behave independently. If such an SoS is to be dependable, it is necessary to ensure that all the constituents are capable of fulfilling their guarantees. We give a mathematically rigorous account of contracts and system models, so that we can verify their conformance.

The design-by-contract paradigm was originally by Meyer [33], and it needs adaptation to express SoS contracts. For SoSs, constituent system designers need to define formal, precise, and verifiable interface specifications for constituents with preconditions, postconditions, and invariants. The postcondition answers the question, "What can the client expect?" The precondition answers the question, "What can the supplier assume?" The invariant answers the question, "What is persistent?" These specifications can then act as contracts which inform the conditions and obligations imposed on a given constituent system.

Parnas calls for formal methods to be "really rethought" [38]; an example he gives is in rethinking the role of termination. Normally, we require programs to terminate to be considered (totally) correct; an extension of this is partial correctness, where it is only required that, if the program terminates, then the answer is correct; but many programs are designed to run indefinitely—specifically reactive systems and particularly SoS. Parnas draws the conclusion that the assertional technique is currently inadequate and that we need to find a good way to represent normal nontermination in correctness arguments.

CML extends the notion of a contract to language constructs not often dealt with in the design-by-contract paradigm, including the use of communication channels, parallel processes, and processes that run forever (addressing Parnas's concerns directly). In fact, every process construct in CML has an associated contract, and this allows clients and suppliers to check that dynamic behaviour conforms to interface specifications.

In the COMPASS project, CML is central to our approach to SoS engineering [17]. We base the approach on a combination of the Systems Modelling Language (SysML) with CML. The former brings facilities for describing system architecture and functionality in a largely graphical, multi-view modelling environment where contracts between constituent systems can be specified at a high-level of abstraction [5]. CML provides a formal basis for analysing SoS models based on SysML abstractions, and adds the rich description of data, functionality, and communication. The loose coupling between SysML and CML has enabled the use of well-established tools to construct CML models; these are translated to a CML version that can be subjected to static and dynamic analysis using the Symphony platform and its plug-ins. The viability of the approach has been demonstrated in diverse ways. For example, in COMPASS, SysML and CML, and tools supporting them, have been successfully deployed together to tackle complex problems in, applications such as home automation [4]. Patterns and profiles have been defined to aid SoS modelling for specific problems such as fault modelling [1].

Our contribution in this paper is to formalise an approach to modelling of contracts in CML, drawing on Meyer's approach to design by contract. In Section 2, we give an overview of CML. In Section 3, we describe the method used to define CML's semantics, Hoare and He's Unifying Theories of Programming, a framework for formalisation of heterogeneous semantics which has a mechanised foundation in Isabelle [18]. In Section 4, we go on to give an overview of CML's semantic domains. In Section 5, we give a series of examples of interface contracts for small fragments of CML processes. In Section 6, we give a complete example of a system in which contracts are used to specify emergent properties. Finally, in Section 7 we reflect on what has been achieved.

2 The COMPASS Modelling Language

The COMPASS Modelling Language (CML) has been developed for the modelling and analysis of Systems of Systems (SoSs), which typically are large-scale systems composed of independent constituent systems [46][1]. CML is based on a combination of VDM [28,16] and CSP [24,42], in the spirit of *Circus* [44,35,36].

A CML model consists of a collection of types, functions, channels, and processes. The type system of CML is taken directly from VDM and includes support for numeric types, finite sets, finite maps, and records, all of which can be further constrained through type invariants. Functions are pure mathematical mappings between inputs and outputs, which can be specified explicitly via λ-calculus, or implicitly using pre and postconditions. In general we adopt the syntax of VDM-SL [14] for the functional and imperative parts of the language, whilst using *Circus* style syntax for the concurrent and reactive parts.

Channels and processes are used to model SoSs, their constituent systems, and the components of these systems. A channel is a medium though which processes can communicate with each other, optionally carrying data of a certain type. Each process encapsulates a number of local state variables, VDM style operations acting on the state (explicit or implicitly specified), and actions which specify the reactive behaviour. Actions are defined using CSP style process constructs, which enable a process to interact with its environment via synchronous communications. The main action constructs of the basic CML language with state, concurrency, and timing are described in Table 1. In addition to the standard CSP operators, such as prefix `a?v -> P` and external choice `P [] Q`, CML includes support for modelling real-time behaviour such as timeout `P [(n)> Q` which will behave like `Q` after `n` time units if `P` does not first interrupt.

Table 1. The core of the CML language

deadlock	`Stop`	*termination*	`Skip`
divergence	`Chaos`	*assignment*	`(v := e)`
specification statement	`[frame w pre p post q]`		
simple prefix	`a -> Skip`	*prefixed action*	`a -> P`
guarded action	`[g] & P`	*sequential composition*	`P ; Q`
internal choice	`P |~| Q`	*external choice*	`P [] Q`
parallel composition	`P [|cs|] Q`	*interleaving*	`P ||| Q`
abstraction	`P \\ A`	*recursion*	`mu X @ P(X)`
wait	`Wait(n)`	*timeout*	`P [(n)> Q`
untimed timeout	`P [> Q`	*interrupt*	`P /\ Q`
timed interrupt	`P /(n)\ Q`	*starts by*	`P startsby(n)`
ends by	`P endsby(n)`	*while*	`while b do P`

An example CML process is shown in Figure 1, which models a stack. We first define the `Element` type for stacks, which in this case are integers constrained to values between 0 and 255 (i.e. bytes) through a type invariant. Next we define five channels with which to communicate with the stack, including an initialisation channel (`init`), channels which indicate whether the stack is empty or not (`empty`, `nonempty`), and finally channels to push and pop, which carry elements.

[1] The COMPASS project is described in detail at `http://www.compass-research.eu`.

The actual Stack process consists of a single state variable stack storing a sequence of elements. We define two operations, Push and Pop, contractually in terms of their pre and postconditions, which respectively add and remove an element from the stack. Push has no precondition, and has the postcondition that the new stack must have the pushed value at its head, and the previous value of the stack (denoted by a tilde) at its tail. Pop requires that that the stack be nonempty, returns the head of the stack, and suitably updates the state. We also define a simple function to check for emptiness of the stack.

```
types
  Element = int inv x == x >= 0 and x <= 255

channels
  init, empty, nonempty
  push, pop : Element

process Stack =
begin
  state stack : seq of Element

  operations
    Push(e : Element)
      post hd(stack) = e and tl(stack) = stack~

    Pop() e : Element
      pre stack <> []
      post stack = tl(stack~) and e = hd(stack)

  functions
    isEmpty : seq of Element -> bool
    isEmpty(s) == s = []

  actions
    Cycle =
      (  push?e -> Push(e)
      [] [not isEmpty(stack)] &
           (dcl v : Element @ v := Pop() ; pop!v -> Skip)
      [] [isEmpty(stack)] & empty -> Skip
      [] [not isEmpty(stack)] & nonempty -> Skip ) ; Cycle

  @ init -> stack := []; Cycle
end
```

Fig. 1. CML model of a stack

The main reactive cycle of the process is defined through the action Cycle, that consists of an external choice between four options. The options describe the following behaviour, respectively:

- wait for input over the `push` channel, and then call the `Push` operation;
- if the stack is not empty, create a local variable `v`, pop the top of the stack and assign it to `v`, and then offer this value over the `pop` channel;
- if the stack is empty, offer communication on the `empty` channel;
- if the stack is not empty, offer communication on the `nonempty` channel.

After each of these behaviours, `Cycle` recurses. The top-level behaviour of the process is then given by the main action, defined after the @ symbol. It waits for an input on `init`, empties the stack, and then enters `Cycle`.

Development of CML models is facilitated through *Symphony*[2], an Eclipse-based integrated development environment. Symphony provides a parser, type checker, simulator, model checker, and theorem prover for CML, all of which have been implemented based on a common semantic basis in the UTP.

The semantics of CML is specification oriented: there is a natural notion of contract for every process language construct and an intuitive refinement ordering. We next describe both notions and give examples of their use.

3 Unifying Theories of Programming

The semantics of CML is defined in Hoare & He's Unifying Theories of Programming (UTP), which is a long-term research agenda for computer science and software engineering [25,11,45]. It can be described as follows: researchers propose programming theories and practitioners use pragmatic programming paradigms; what is the relationship between them? UTP, based on predicative programming [23], gives three principal ways to study such relationships: (i) by computational paradigm, identifying common concepts; (ii) by level of abstraction, from requirements, through architectures and components, to platform-specific implementation technologies; and (iii) by method of presentation—namely, denotational, algebraic, and operational semantics—and their mutual embeddings.

UTP presents a theoretical foundation for understanding software and systems engineering. In its original presentation, it describes nondeterministic sequential programming, the refinement calculus, the algebra of programming, compilation of high-level languages, concurrency, communication, reactive processes, and higher-order programming [25]. Subsequently, it has been exploited in a diversity of areas such as component-based systems [49], hardware verification [40,39], and hardware/software co-design [2].

UTP can also be used in a more active way for constructing domain-specific languages, especially ones with heterogeneous semantics. Examples include the semantics for *Circus* [35,36,44] and Safety-Critical Java (SCJ) [8,10,9], both of which have been composed from individual, reusable theories of sequential and concurrent programming. SCJ additionally has real-time tasking and a sophisticated model of memory usage. The analogy for these kinds of compositional semantics is of a theory supermarket, where you shop for exactly those features you need, whilst being confident that the theories plug-and-play nicely together.

[2] Symphony can be obtained from `http://symphonytool.org/`

The semantic metalanguage for UTP is an alphabetised version of Tarski's relational calculus, presented in a pointwise predicative style that is reminiscent of the schema calculus in the Z notation [47]. Each programming construct is formalised as a relation between an initial and an intermediate or final observation. The collection of these relations forms a *theory* of a paradigm that contains three essential parts: an alphabet, a signature, and some healthiness conditions.

The alphabet is a set of variable names that gives the vocabulary for the theory being studied: names are chosen for any relevant external observations of behaviour. For instance, programming variables x, y, and z are normally part of the alphabet. Theories for particular programming paradigms require the observation of extra information. Some examples from existing theories are: a flag that says whether the program has started (ok); the current time ($clock$); the number of available resources (res); a trace of the events in the life of the program (tr); a set of refused events (ref); or a flag that says whether the program is waiting for interaction with its environment ($wait$).

The signature gives syntactic rules for denoting objects of the theory. Finally, healthiness conditions identify properties that characterise the predicates of the theory. They are often expressed in terms of a function ϕ that makes a program healthy. There is no point in applying ϕ twice, since we cannot make a healthy program even healthier, so ϕ must be idempotent: $P = \phi(\phi(P))$. The fixed-points of this equation are the healthy predicates of the theory.

An alphabetised predicate ($P, Q, \ldots,$ ***true***) is an alphabet-predicate pair, such that the predicate's free variables are all members of the alphabet. Relations are predicates in which the alphabet comprises plain variables (x, y, z, ...) and dashed variables (x', a', ...); the former represent initial observations, and the latter, intermediate or final observations. The alphabet of P is denoted αP. A *condition* ($b, c, d, \ldots, true$) has no dashed variables. Predicate calculus operators combine predicates in the obvious way, but with specified restrictions on the alphabets of the operands and specified resulting alphabet. For example, the alphabet of a conjunction is the union of the alphabets of its components, and disjunction is defined only for predicates with the same alphabet.

A distinguishing feature of UTP is the central role played by program correctness, which is defined in the same way in every programming paradigm in [25]. Informally, it is required that, in every state, the behaviour of an implementation implies its specification. If we suppose that for a predicate P, $\alpha P = \{a, b, a', b'\}$, then the universal closure of P is $\forall\, a, b, a', b' \bullet P$, denoted $[\, P\,]$. P is correct with respect to a specification S providing every observation of P is also an observation of S: $[\, P \Rightarrow S\,]$; this is described by $S \sqsubseteq P$ (read S is refined by P).

UTP has an infix syntax for the conditional, $P \lhd b \rhd Q$, and it is defined as $(b \wedge P) \vee (\neg\ b \wedge Q)$, if $\alpha b \subseteq \alpha P = \alpha Q$. Sequence is modelled as relational composition: two relations may be composed, providing that their alphabets match: $P(v')\ ;\ Q(v)\ \widehat{=}\ \exists\, v_0 \bullet P(v_0) \wedge Q(v_0)$, if $out\alpha P = in\alpha Q' = \{v'\}$. If $A = \{x, y, \ldots, z\}$ and $\alpha e \subseteq A$, then the assignment $x :=_A e$ defined below, with expression vector e and variable vector x, changes only x's value.

$$x :=_A e\ \widehat{=}\ (x' = e \wedge y' = y \wedge \cdots \wedge z' = z)$$

There is a degenerate form of assignment that changes no variable; it is called "skip" defined as $\mathit{II}_A \mathrel{\widehat{=}} (v' = v)$, if $A = \{v\}$. Nondeterminism can arise in one of two ways: either as the result of runtime factors, such as distributed processing or as the under-specification of implementation choices. Either way, nondeterminism is modelled by choice; the semantics is disjunction: $P \sqcap Q \mathrel{\widehat{=}} P \lor Q$.

The set of alphabetised predicates with a particular alphabet A forms a complete lattice under the refinement ordering (which is a partial order). The bottom element is denoted $\bot_{\mathrm{A}}$, and is the weakest predicate ***true***; this is the program that aborts, and behaves quite arbitrarily. The top element is denoted $\top^A$, and is the strongest predicate ***false***; this is the program that performs miracles and implements every specification. Since alphabetised relations form a complete lattice, every construction defined solely using monotonic operators has a complete lattice of fixed points. The weakest fixed-point of the function F is denoted by μF, and is simply the greatest lower bound (the *weakest*) of all the fixed-points of F. This is defined: $\mu F \mathrel{\widehat{=}} \bigsqcap\{\, X \mid F(X) \sqsubseteq X \,\}$. The strongest fixed-point νF is the dual of the weakest fixed-point.

4 CML Semantics

Currently, CML contains several language paradigms. These are all presented through formalisation as UTP theories.

State-based description. The theory of designs provides a nondeterministic programming language with precondition and postcondition specifications as contracts, written $P \vdash Q$, for precondition P and postcondition Q. The concrete realisation is VDM.

Concurrency and communication. The theory of reactive processes provides process networks communicating by message passing. The concrete realisation is CSP_M with its rich collection of process combinators.

Object orientation. This theory is built on designs with state-based descriptions structured by sub-typing, inheritance, and dynamic binding, with object creation, type testing and casting, and state-component access [7].

References. The theory of heap storage and its manipulations supports a reference semantics based on separation logic.

Time. The theory of timed traces in UTP supports the observation of events in discrete time. It is used in a theory of Timed CSP.

As explained in the previous section, the semantic domains are each formalised as lattices of relations ordered by refinement. Mappings exist between the different semantic domains that can be used to translate a model from one lattice into a corresponding model in another lattice. For example, the lattice of designs is completely disjoint from the lattice of reactive processes, but a mapping ***R*** maps every design into a corresponding reactive process. Intuitively, the mapping equips the design with the crucial properties of a reactive process: that it has a trace variable (tr) that records the history of interactions with its environment and that it can wait for such interactions. A vital healthiness condition is that this trace increases monotonically: this ensures that once an event has taken place it cannot be retracted—even when the process aborts.

Another mapping counteracts $\boldsymbol{R}$: it is called $\boldsymbol{H}$, and it is the function that characterises what it is to be a design. $\boldsymbol{H}$ puts requirements on the use of the observations ok and ok', and it is the former that concerns us here. It states that, until the operation has started properly (ok is true), no observation can be made of the operation's behaviour. So, if the operation's predecessor has aborted, nothing can be said about any of the operation's variables, not even the trace observation variable. This destroys the requirement of $\boldsymbol{R}$ that restricts the trace so that it only ever increases monotonically, even when ok is false.

This pair of mappings forms a Galois connection [13], and such pairs exist between all of CML's semantic domains. One purpose of a Galois connection is to embed one theory within another, and this is what gives the compositional flavour of UTP and CML, since Galois connections compose to form other Galois connections. For example, if we establish a Galois connection between reactive processes and timed reactive processes, then we can compose the connection between designs and reactive processes with this new Galois connection to form a connection between designs and timed reactive processes.

The possibly obscure mathematical fact that there is a Galois connection between designs and reactive processes is of great practical value. One of the most important features of designs is assertional reasoning based on preconditions and postconditions, including the use of Hoare logic and weakest precondition calculus. Assertional reasoning, as defined in the theory of designs, can be incorporated into the theory of reactive processes by means of the mapping $\boldsymbol{R}$.

In the theory of designs, a Hoare triple $\{p\}\ Q\ \{r\}$, where p is a precondition, r is a postcondition, and Q is a reactive process, is given the meaning $(\boldsymbol{R}(p \vdash r') \sqsubseteq Q)$, which is a refinement assertion. In the specification $\boldsymbol{R}(p \vdash r')$ the precondition p and the postcondition r are assembled into a design, with r as a condition on the after-state; this design is then translated into a reactive process using $\boldsymbol{R}$. The semantics of the Hoare triple requires that this reactive specification is implemented correctly by the reactive process Q. Thus, reasoning with preconditions and postconditions is, in this way, extended from the state-based operations of the theory of designs to cover all operators of the reactive language, including non-terminating processes, concurrency, and communication.

This is the foundation of the contractual approach used in COMPASS: preconditions and postconditions (designs) are embedded in each of the semantic domains and this brings uniformity through a familiar reasoning technique.

5 Contracts in CML

In this section, we give a series of examples of the use of contracts in CML.

Example 1 (Single shot). The CML action $a \rightarrow \mathit{Skip}$ will perform one a event and then terminate. It never diverges, so it has precondition ***true***. Its postcondition depends on whether or not it is waiting, indicated by the observational variable $wait'$ being true. If it is waiting, then it has not performed any events and the trace is unchanged ($tr' = tr$), but it is also not refusing to perform the a event: $a \notin ref'$, where ref' is the refusal set. Otherwise, it has performed exactly one a event ($tr' = tr ^\frown \langle a \rangle$). This precondition-postcondition pair forms a design that gives the contract for the action; of course, the contract must also insist on $\mathbf{R}$-healthiness. In full, the contract is as follows.

$$\mathbf{R}(\mathbf{true} \vdash (tr' = tr \wedge a \notin ref') \lhd wait' \rhd tr' = tr ^\frown \langle a \rangle)$$

We notice the use of observational variables: ok, ok', $wait$, $wait'$, tr, tr', ref, and ref'. These are "ghost variables", not code; that is, they are part of the underlying semantic model and cannot be manipulated at run time. Ghost variables provide a convenient way of forming contracts by allowing us explicitly to restrict possible reactive behaviours.

There is a technicality about any assertion involving the ghost variable ref'. If an action may refuse a set s, then it may refuse any subset of s. That is, if an action refuses the set $\{a, b\}$, then it will also refuse the sets $\{a\}$ and $\varnothing$, for example. For this reason, an assertion such as $a \in ref'$ is unsatisfiable. So if we really did want to assert that a is always refused, then we would instead say that it never occurs: it never appears in the trace, rather than restrict refusals.

The precise form of a CML contract is derived from the fact that every CML action can be expressed in the form $\mathbf{R}(P \vdash Q)$. We saw the syntactic form of a design above; its semantics depends on the two observations mentioned in Section 4, ok and ok': if the design is started (ok is true) in a state in which the precondition holds (P is true), then it must terminate (ok' will be true) and when it does, the postcondition must hold (Q must be true). This justifies the definition $P \vdash Q \ = \ ok \wedge P \Rightarrow ok' \wedge Q$ for designs.

Example 2 (Chocolate vending machine). We consider a grossly simplified model of a vending machine `VM`. A complete transaction with the machine involves inserting a coin and extracting a chocolate; the machine repeatedly engages in such transactions as specified by the action below.

```
VM = coin -> choc -> VM
```

This is a CML action, but what is the contract? We notice that there is no state, so the contract must be entirely in terms of the ghost variables ok, ok', $wait$, $wait'$, tr, tr', ref, and ref'. A reasonable contract for the machine comes in two parts: a requirement on the user and a requirement on the machine itself.

– The machine should not lose money: *every chocolate must be paid for.*

```
NOLOSS = freq(choc,tr'-tr) <= freq(coin,tr'-tr)
```

– The machine should be fair: *the machine should not build up too much credit.*

```
FAIR = freq(coin,tr'-tr) <= freq(choc,tr'-tr) + 1
```

The auxiliary function `freq` gives the frequency of an event in a trace. It is defined below. The specification of the vending machine is given by the conjunction of these requirements. It is defined below.

```
VMSPEC = NOLOSS and FAIR
```

The VDM (and, therefore, CML) definition of `freq` is as follows.

```
freq: Event * (seq of Event) -> nat
freq(e,s) =
  if s = [] then
    0
  else
    if hd(s) = e then
      1 + freq(e,tl(s))
    else
      freq(e,tl(s))
```

How can we check that `VM` satisfies this specification? There are four principal ways, two using theorem proving and two using model checking:

1. prove that $[\, VM \Rightarrow VMSPEC \,]$,
2. use the assertional technique (that is, Hoare logic),
3. use a refinement model checker, or
4. use a temporal logic model checker.

(1), (2), and (3) are essentially the same: they check the refinement relation. We may regard (1) as a full-frontal attack on the problem using a theorem prover. On the other hand, (2) is more subtle, using inference rules to match the structure of the implementation `VM` and check the assertion `VMSPEC`. For (3), the specification must be captured as a finite state CML action, just like the implementation. A model checker (such as [21] or [31]) is then used to check that the observable behaviours of the implementation are all behaviours of the specification.

For (4), the specification must be captured as an expression in temporal logic; LTL is commonly used [3], with its operators such as "eventually" and "henceforth". A model checker, such as PAT [31] or the CML model checker [34], is then used to test whether the system satisfies the temporal logic specification. Roscoe [43] and Lowe [32] have each studied the relationship between refinement-based checking and temporal-logic checking. An account of that in the UTP can justify its use for CML.

To illustrate the use of the refinement model checker, we construct actions to embody the two parts of the specification. First, for `NOLOSS`, we construct an action parametrised by the number of coins and chocolates already dispensed. `NOLOSS` is always willing to accept further coins, since that cannot contribute to a financial loss, but dispenses a chocolate only if there is outstanding credit.

```
NoLossProc =
  coins, chocs: nat @
    coin -> NoLossProc(coins+1,chocs)
    []
    [chocs < coins] & choc -> NoLossProc(coins,chocs+1)
```

The action embodying `FAIR` complements `NOLOSS`: it is always willing to dispense chocolates, since that cannot be unfair to a customer, but accepts a coin only if at least as many chocolates have been dispensed as paid for.

```
FairProc = coins, chocs: nat @
  [coins <= chocs] & coin -> NoLossProc(coins+1,chocs)
  []
  choc -> FairProc(coins,chocs+1)
```

A vending machine that does nothing makes no loss and is trivially fair, so `NOLOSS and FAIR` is not a very good specification; so how do we say something stronger? There is a liveness aspect to fairness: if the customer has paid for a chocolate, then the machine should not refuse to dispense it.

```
FAIR1 = (freq(choc,tr'-tr) < freq(coin,tr'-tr) => choc not in ref')
```

Similarly, there is a liveness aspect to profit making: if every chocolate that has been paid for has been dispensed, then the machine should not refuse a coin.

```
PROFIT1 = (freq(choc,tr'-tr) = freq(coin,tr'-tr) => coin not in ref')
```

The two actions embodying our specification already have these two properties.

Specifications involving *ref′* can be used to assert deadlock freedom. If the event alphabet for an action is A, then deadlock freedom can be specified as `NONSTOP = ref' <> A`. This states that the action can never reject, that is, refuse, the entire alphabet of events, and so is never deadlocked.

When checking with a model checker such as FDR[3], it is sufficient to check satisfaction of each part of the specification independently. So to check that the vending machine does not make a loss and that it is fair to its customers, we can use the two separate assertions below.

```
assert NoLossProc [= VM

assert FairProc [= VM
```

Equivalently, we can check the two properties simultaneously. To do this, we need to assemble the `NoLossProc` and `FairProc` actions in parallel, synchronising on the `choc` and `coin` events as shown below.

```
assert NoLossProc [|{choc, coin}|] FairProc [= VM
```

The process *Chaos* misbehaves badly, like your worst nightmare: it melts down the reactor; it switches off the in-flight computer; it transfers all your funds into my bank account. The next examples illustrate its use.

Example 3 (Deferred chaos). Consider the process:

```
a -> Chaos
```

[3] `http://www.fsel.com`.

This process is perfectly safe, providing its environment never engages in the event a. So what is the contract? The precondition must record the assumption that the a event never occurs, which it does as a relation: $\neg\,(tr \frown \langle a \rangle \le tr')$. The precondition is describing a protocol in terms of the trace, a kind of rely-condition in the sense of Jones [27]. Now, if we assume that the precondition holds, then the postcondition is straightforward: the action is forever waiting ($wait'$), and the trace never changes ($tr' = tr$), but a is not refused ($a \notin ref'$).

$$\boldsymbol{R}(\neg\,(tr \frown \langle a \rangle \le tr') \vdash wait' \wedge tr' = tr \wedge a \notin ref')$$

Example 4 (Badly behaved vending machine). We now consider the following badly-behaved vending machine `VMC`.

```
VMC =
  in2 -> ( large -> VMC
           []
           small -> out1 -> VMC )
  []
  in1 -> ( small -> VMC
           []
           in1 -> ( large -> VMC
                    []
                    in1 -> Chaos ) )
```

Initially, `VMC` is prepared to accept either a £1 coin or a £2 coin. If the £2 coin is inserted, then the customer has a choice between a large and a small chocolate bar. If the small bar is selected, then the machine offers change of £1. Alternatively, if the £1 coin is inserted, then a choice is offered between extracting a small chocolate bar or inserting a further £1 coin. If another £1 coin is inserted, the choice then becomes between extracting a large chocolate bar or inserting yet another £1 coin, whereupon the machine behaves chaotically. The precondition here is, therefore, $\neg\,(tr \frown [in1, in1, in1] \le tr')$.

6 Mini-Mondex

Mondex[4] is an electronic purse hosted on a smart card and developed about fifteen years ago to the high-assurance standard ITSEC Level E6 [26] by a consortium led by NatWest, a UK high-street bank. Eight years ago, a community effort was launched to mechanically verify the original models of Mondex in a variety of different notations, in order to compare and contrast their effectiveness [48,41,22,6,30,20,19]; the problem has now become a benchmark for formal verification. In this section, we describe a simplified version of the problem: mini-Mondex, where we ignore faulty behaviour and focus on specifying functional requirements.

[4] `http://www.mondex.com`.

Purses interact using a communications device, and strong guarantees are needed that transactions are secure in spite of power failures and mischievous attacks. These guarantees ensure that electronic cash cannot be counterfeited, although transactions are completely distributed. There is no centralised control: all security measures are locally implemented, with no real-time external audit logging or monitoring; key properties *emerge* from local behaviour.

Our model of Mondex has the following constant values: `N`, the number of cards in the system; `V`, the maximum value that may be held by a card; and `M`, the total money supply. These are specified in CML as follows.

```
values
  N: nat = 10
  V: nat = 10
  M: nat = N*V
```

There are two types related to these constants: the `Index` set for cards; and the `Money`. The relationship with `N` and `M` is made explicit through two invariants.

```
types
  Index = nat
  inv i == i in set {1,...,N}

  Money = nat
  inv m == m in set {0,...,M}
```

We also specify three functions which are needed for our contract. `initseq(n)` builds a sequence of numbers from 0 to `n`. `subtseq(xs, i, n)` subtracts `n` from the `i`th item of sequence `xs`. `addseq(xs, i, n)` adds `n` to the `i`th item of `xs`.

```
functions
  initseq: nat -> seq of nat
  initseq(n) == [i | i in set {0,...,n}]

  subtseq: seq of nat * nat * nat -> seq of nat
  subtseq(xs, i, n) == xs ++ {i |-> xs(i) - n}
    pre len(xs) > i and xs(i) >= n

  addseq: seq of nat * nat * nat -> seq of nat
  addseq(xs, i, n) == xs ++ {i |-> xs(i) + n}
    pre len(xs) > i
```

There are a number of channels that connect cards with each other and with the environment. A user can instruct one card to pay another with the event `pay.i.j.n`, which corresponds to instructing card `i` to pay card `j` the sum of `n` money units. The attempted transfer of money is made between cards using the `transfer` channel. The transaction may be `accepted` or `rejected`.

```
channels
  pay, transfer: Index * Index * Money
  accept, reject: Index
```

Each card is modelled by an indexed CML process with its encapsulated state. The `state` of the process consists of a single component `value`, which is a natural number recording the balance in the purse. There are three operations: (i) `Init`, which sets the initial value to `V`; (ii) `Credit`, which increments the `value` by the parameter `n`; (iii) and `Debit`, which decrements the `value` by the parameter `n`. There are also three actions: (i) `Transfer` accepts a `pay` communication and analyses it to see if there are sufficient funds to honour the debit, replying appropriately with an `accept` or `reject` communication; if there are sufficient funds, then a `transfer` message is sent to the receiving card and debits its local state. (ii) `Receive` accepts a `receive` message and credits its local state appropriately. (iii) `Cycle` repeatedly offers the `Transfer` and `Receive` actions. The `@`-symbol marks the main action for the process.

```
process Card = val i: Index @
  begin
    state value: nat
    operations
      Init: () ==> ()
      Init() == value := V

      Credit: nat ==> ()
      Credit(n) == value := value + n

      Debit: nat ==> ()
      Debit(n) == value := value - n
      pre n <= value

    actions
      Transfer =
        pay.i?j?n ->
          ( [n > value] & reject!i -> Skip
            []
            [n <= value] & transfer.i.j!n -> accept!i -> Debit(n) )

      Receive = transfer?j.i?n -> Credit(n)

      Cycle = ( Transfer [] Receive ); Cycle
  @
      Init(); Cycle
  end
```

The network is defined by the parallel composition of all the indexed cards. We need to specify the interface for each card to specify its interaction with the rest of the network. As defined above, `Card(i)` participates in the following events.

– Every event of the form `pay.i.j.n`, for any card `j` and amount `n`.
– Every event of the form `transfer.i.j.n`, for any card `j` and amount `n`. These represent the money leaving the card.

– Every event of the form `transfer.j.i.n`, for any card `j` and amount `n`. These represent the money entering the card.
– The events `accept.i` and `reject.i`.

In the construction of the network, we identify this alphabet of events for each of the `Card(i)` processes, and assemble the `N` cards in parallel.

```
process Cards =
  || i in set {1,...,N} @
    [{| pay.i,transfer.i,accept.i, reject.i|} union
      {| transfer.j.i.n | j in set {1,...,N}, n in set {0,...,M}|}
    ] Card(i)
```

Cards that share the same event in their alphabet need to synchronise on that event. The network, therefore, ensures that transfers between cards `i` and `j` are achieved when both cards cooperate.

Finally, we need to hide the internal channels that connect the cards: these do not form part of the extensional behaviour of the network:

```
process Network = Cards \ {|transfer|}
```

We identify the following properties that are required of mini-Mondex.

No counterfeiting: There must be no increase in the total value in the system.
Fairness: There must be no loss of value.
Usefulness: If we demand a transfer from a card that has the required funds, then the transfer should take place.

We notice that the first two properties above are emergent global properties, but the system has only local behaviour. In what follows, we describe these properties using the CML contract `Spec` below.

This contract models the network as a single process with an Olympian view of the state of all cards; it has only one state component, `valueseq`, which is a sequence of numbers, indexed by card indexes. An invariant requires there to be an element in the sequence for every card. The `valueseq` is initialised to correspond with the initialisation of each card: each containing the value `V`.

```
process Spec =
  begin
    state
      valueseq: seq of nat
    inv
      len(valueseq) = N
    operations
      Init: () ==> ()
      Init() == valueseq := initseq(N)
```

There are two actions. The first, `Pay` is parametrised by source and destination cards and an amount to be paid. Its behaviour starts with the communication `pay.i.j.n`. Following this, there is an analysis of whether the card paying the amount can afford it. If it cannot, then the transaction is rejected. If it can, then the payer's and payee's values are updated accordingly to reflect the transfer of money, and the transaction is accepted.

The second action is a repetitive cycle; on each step, a nondeterministic choice is made of a payer, a payee, and an amount to be paid. Thus, `Cycle` represents all possible financially correct transactions.

```
    actions
      Pay = i,j: Index, n: Money @
        pay.i.j.n ->
          if n > valueseq(i) then
            reject.i -> Skip
          else
            ( valueseq := subtseq(valueseq,i,n);
              valueseq := addseq(valueseq,j,n);
              accept.i -> Skip )

      Cycle =
        ( |~| i,j: Index, n: Money @ Pay(i,j,n) );
        Cycle
  @
    Cycle
  end
```

`Spec` gives us an arena in which to specify the correctness of mini-Mondex. The properties identified above can be described as follows.

No increase in value: `sum(valueseq) <= M`. (`sum` returns the sum of the elements in a sequence.)

No loss in value: `sum(valueseq) >= M`.

Usefulness If we demand a transfer, and we have got the funds, then the transfer should take place:

```
    forall i, j: Index; n: Money @
      tr'-tr <> []
      and last(tr'-tr) = transfer.i.j.n
      and n >= valueseq(i)
        =>
          accept.i not in ref'
```

Finally, we need an invariant that relates the stored state, `valueseq`, to the history of transactions. For card `i`, the communications `transfer.i.j` represent outgoing payments; the communications `transfer.j.i` represent incoming payments; and the value `V` represents the initial value in the card.

```
forall i: {1,...,N} @
  valueseq(i) =
    V + transum((tr'-tr) filter { transfer.i.j | j in set {1,...,N} })
      + transum((tr'-tr) filter { transfer.j.i | j in set {1,...,N} })
```

where

```
transum(s) =
  if s = [] then
    0
  else
    amount(hd(s)) + transum(tl(s))

forall i, j: Index; n: Money @ amount(transum.i.j.n) = n
```

7 Conclusions

We have presented a series of examples of the use of contracts in CML. It is based on the semantic embedding of a theory of total correctness based on preconditions and postconditions into the theory that defines the semantic model of CML. With that, we have a characterisation of preconditions and postconditions of reactive constructs, including communication, choice, and parallelism.

De Boer describes the postconditions of nonterminating processes as equivalent to *false* [15], since he considers their states to be unobservable; this is of little use in reasoning about reactive processes that run forever. As we mentioned in Section 1, Parnas calls for the development of assertional techniques to handle the normal nontermination of reactive processes [38]. Our work explicitly considers stable intermediate states (those satisfying the ghost expression $ok' \wedge wait'$), and provides these states with postconditions.

Jones has defined rely and guarantee conditions for assertional reasoning [27] in the presence of concurrency. These conditions are concerned with properties of interleaved atomic steps: guarantee conditions describe postconditions for atomic steps of the process, and rely conditions describe postconditions for atomic steps of the environment. Both rely and guarantee conditions are relations and, therefore, regarded as postconditions. Our postconditions are analogous to guarantee conditions, except that they relate initial states to intermediate states, rather than describing the postcondition of an arbitrary atomic step. Similarly, our preconditions are relational and are analogous to Jones's rely conditions, except that, again, they relate initial and intermediate states, rather than describing the postcondition of an atomic step of the environment.

Future work includes exploring the relationship between our contractual techniques and Jones's atomic-step semantics for rely and guarantee thinking. The development of tools to support assertional reasoning in CML is also essential for its practical relevance and scalability. Symphony does not yet support ghost variables.

The CML semantics has been partially mechanised in Isabelle, through a semantic embedding of UTP called Isabelle/UTP [18]. In particular we have mechanised the theory of designs, the theory of reactive processes, and have preliminary support for a Hoare calculus, which together provide the building blocks for formal verification of contracts as shown in this paper. We have already used Isabelle/UTP to construct a CML theorem prover [12], and a verification-condition generator is a natural next step in this effort. Moreover we are currently working on a refinement tool for Symphony which provides calculational support for the CML refinement calculus, building on previous work [37]. This will provide tool support for showing conformance between a given CML contract and an underlying SoS, such as the Mondex example.

References

1. Andrews, Z., Fitzgerald, J., Payne, R., Romanovsky, A.: Fault Modelling for Systems of Systems. In: Proceedings of the 11th International Symposium on Autonomous Decentralised Systems (ISADS 2013), pp. 59–66 (March 2013)
2. Beg, A., Butterfield, A.: Linking a state-rich process algebra to a state-free algebra to verify software/hardware implementation. In: FIT 2010, 8th Intl Conf. on Frontiers of Information Technology, Islamabad, p. 47. ACM (2010)
3. Ben-Ari, M., Manna, Z., Pnueli, A.: The temporal logic of branching time. In: White, J., Lipton, R.J., Goldberg, P.C. (eds.) 8th Ann. ACM Symp. on Principles of Programming Languages, Williamsburg, pp. 164–176. ACM Press (1981)
4. Bryans, J., Fitzgerald, J., Payne, R., Kristensen, K.: Maintaining emergence in systems of systems integration: a contractual approach using SysML. In: INCOSE International Symposium (to appear, 2014)
5. Bryans, J., Fitzgerald, J., Payne, R., Miyazawa, A., Kristensen, K.: SysML Contracts for Systems of Systems. In: 9th Intl Conf. on Systems of Systems Engineering (SoSE). IEEE (June 2014)
6. Butler, M., Yadav, D.: An incremental development of the Mondex system in Event-B. Formal Asp. Comput. 20(1), 61–77 (2008)
7. Cavalcanti, A., Sampaio, A., Woodcock, J.: Unifying classes and processes. Software and System Modeling 4(3), 277–296 (2005)
8. Cavalcanti, A., Wellings, A., Woodcock, J.: The Safety-Critical Java memory model: A formal account. In: Butler, M., Schulte, W. (eds.) FM 2011. LNCS, vol. 6664, pp. 246–261. Springer, Heidelberg (2011)
9. Cavalcanti, A., Wellings, A.J., Woodcock, J.: The Safety-Critical Java memory model formalised. Formal Asp. Comput. 25(1), 37–57 (2013)
10. Cavalcanti, A., Wellings, A.J., Woodcock, J., Wei, K., Zeyda, F.: Safety-critical Java in Circus. In: Wellings, A.J., Ravn, A.P. (eds.) The 9th Intl Workshop on Java Technologies for Real-time and Embedded Systems, JTRES 2011, York, pp. 20–29. ACM (2011)
11. Cavalcanti, A., Woodcock, J.: A tutorial introduction to CSP in Unifying Theories of Programming. In: Cavalcanti, A., Sampaio, A., Woodcock, J. (eds.) PSSE 2004. LNCS, vol. 3167, pp. 220–268. Springer, Heidelberg (2006)
12. Couto, L., Foster, S., Payne, R.: Towards verification of constituent systems through automated proof. In: Proc. Workshop on Engineering Dependable Systems of Systems (EDSoS). ACM CoRR (2014)
13. Davey, B.A., Priestley, H.A.: Introduction to Lattices and Order, 2nd edn. Cambridge University Press (2002)
14. Dawes, J.: The VDM-SL Reference Guide. Pitman (1991) ISBN 0-273-03151-1
15. de Boer, F.S., Hannemann, U., de Roever, W.-P.: Hoare-style compositional proof systems for reactive shared variable concurrency. In: Ramesh, S., Sivakumar, G. (eds.) FST TCS 1997. LNCS, vol. 1346, pp. 267–283. Springer, Heidelberg (1997)

16. Fitzgerald, J., Larsen, P.G., Mukherjee, P., Plat, N., Verhoef, M.: Validated Designs for Object-oriented Systems. Springer (2005)
17. Fitzgerald, J., Larsen, P.G., Woodcock, J.: Foundations for Model-based Engineering of Systems of Systems. In: Aiguier, M., et al. (eds.) Complex Systems Design and Management, pp. 1–19. Springer (January 2014)
18. Foster, S., Zeyda, F., Woodcock, J.: Isabelle/UTP: A mechanised theory engineering framework. In: 5th International Symposium on Unifying Theories of Programming (to appear, 2014)
19. Freitas, L., Woodcock, J.: Mechanising Mondex with Z/Eves. Formal Asp. Comput. 20(1), 117–139 (2008)
20. George, C., Haxthausen, A.E.: Specification, proof, and model checking of the Mondex electronic purse using RAISE. Formal Asp. Comput. 20(1), 101–116 (2008)
21. Gibson-Robinson, T., Armstrong, P., Boulgakov, A., Roscoe, A.W.: FDR3 — A modern refinement checker for CSP. In: Ábrahám, E., Havelund, K. (eds.) TACAS 2014. LNCS, vol. 8413, pp. 187–201. Springer, Heidelberg (2014)
22. Haneberg, D., Schellhorn, G., Grandy, H., Reif, W.: Verification of Mondex electronic purses with KIV: from transactions to a security protocol. Formal Asp. Comput. 20(1), 41–59 (2008)
23. Hehner, E.C.R.: Retrospective and prospective for Unifying Theories of Programming. In: Dunne, S., Stoddart, B. (eds.) UTP 2006. LNCS, vol. 4010, pp. 1–17. Springer, Heidelberg (2006)
24. Hoare, C.A.R.: Communicating Sequential Processes. Prentice-Hall (1985)
25. Hoare, C.A.R., He, J.: Unifying Theories of Programming. Prentice Hall (1998)
26. ITSEC. Information Technology Security Evaluation Criteria (ITSEC): Preliminary harmonised criteria. Technical Report Document COM(90) 314, Version 1.2, Commission of the European Communities (1991)
27. Jones, C.B.: Specification and design of (parallel) programs. In: IFIP Congress, pp. 321–332 (1983)
28. Jones, C.B.: Systematic Software Development using VDM, 2nd edn. Prentice Hall International (1990)
29. Kopetz, H.: System-of-Systems complexity. In: Proc. 1st Workshop on Advances in Systems of Systems, pp. 35–39 (2013)
30. Kuhlmann, M., Gogolla, M.: Modeling and validating Mondex scenarios described in UML and OCL with USE. Formal Asp. Comput. 20(1), 79–100 (2008)
31. Liu, Y., Sun, J., Dong, J.S.: PAT 3: An extensible architecture for building multi-domain model checkers. In: Dohi, T., Cukic, B. (eds.) IEEE 22nd Intl Symp. on Software Reliability Engineering, ISSRE 2011, Hiroshima, pp. 190–199. IEEE (2011)
32. Lowe, G.: Specification of communicating processes: temporal logic versus refusals-based refinement. Formal Asp. Comput. 20(3), 277–294 (2008)
33. Meyer, B.: Applying "design by contract". IEEE Computer 25(10), 40–51 (1992)
34. Mota, A., Farias, A., Didier, A., Woodcock, J.: Rapid prototyping of a semantically well founded Circus model checker. In: Giannakopoulou, D., Salaün, G. (eds.) SEFM 2014. LNCS, vol. 8702, pp. 235–249. Springer, Heidelberg (2014)
35. Oliveira, M., Cavalcanti, A., Woodcock, J.: A denotational semantics for Circus. Electr. Notes Theor. Comput. Sci. 187, 107–123 (2007)
36. Oliveira, M., Cavalcanti, A., Woodcock, J.: A UTP semantics for Circus. Formal Asp. Comput. 21(1-2), 3–32 (2009)
37. Oliveira, M., Gurgel, A.C., Castro, C.G.: CRefine: Support for the Circus refinement calculus. In: 6th Intl. Conf. on Software Engineering and Formal Methods (SEFM 2008), pp. 281–290. IEEE Computer Society (November 2008)

38. Parnas, D.L.: Really rethinking 'formal methods'. IEEE Computer 43(1), 28–34 (2010)
39. Perna, J.I., Woodcock, J.: Mechanised wire-wise verification of Handel-C synthesis. Sci. Comput. Program. 77(4), 424–443 (2012)
40. Perna, J.I., Woodcock, J., Sampaio, A., Iyoda, J.: Correct hardware synthesis—an algebraic approach. Acta Inf. 48(7-8), 363–396 (2011)
41. Ramananandro, T.: Mondex, an electronic purse: specification and refinement checks with the Alloy model-finding method. Formal Asp. Comput. 20(1), 21–39 (2008)
42. Roscoe, A.W.: The Theory and Practice of Concurrency. Prentice Hall International (1997)
43. Roscoe, A.W.: On the expressive power of CSP refinement. Formal Asp. Comput. 17(2), 93–112 (2005)
44. Woodcock, J., Cavalcanti, A.: The semantics of Circus. In: Bert, D., Bowen, J.P., Henson, M.C., Robinson, K. (eds.) ZB 2002. LNCS, vol. 2272, pp. 184–203. Springer, Heidelberg (2002)
45. Woodcock, J., Cavalcanti, A.: A tutorial introduction to designs in Unifying Theories of Programming. In: Boiten, E.A., Derrick, J., Smith, G. (eds.) IFM 2004. LNCS, vol. 2999, pp. 40–66. Springer, Heidelberg (2004)
46. Woodcock, J., Cavalcanti, A., Fitzgerald, J.S., Larsen, P.G., Miyazawa, A., Perry, S.: Features of CML: A formal modelling language for systems of systems. In: 7th Intl Conf. on Systems of Systems Engineering, SoSE 2012, Genova, pp. 445–450. IEEE (2012)
47. Woodcock, J., Davies, J.: Using Z: Specification, Refinement, and Proof. Prentice-Hall, Inc. (1996)
48. Woodcock, J., Stepney, S., Cooper, D., Clark, J.A., Jacob, J.: The certification of the Mondex electronic purse to ITSEC Level E6. Formal Asp. Comput. 20(1), 5–19 (2008)
49. Zhan, N., Kang, E.Y., Liu, Z.: Component publications and compositions. In: Butterfield, A. (ed.) UTP 2008. LNCS, vol. 5713, pp. 238–257. Springer, Heidelberg (2010)

Distributed Energy Management Case Study: A Formal Approach to Analyzing Utility Functions

Aida Čaušević, Cristina Seceleanu, and Paul Pettersson

Mälardalen Real-Time Research Centre (MRTC),
Mälardalen University, Västerås, Sweden
{aida.causevic,cristina.seceleanu,paul.pettersson}@mdh.se

Abstract. The service-oriented paradigm has been established to enable quicker development of new applications from already existing services. Service negotiation is a key technique to provide a way of deciding and choosing the most suitable service, out of possibly many services delivering similar functionality but having different response times, resource usages, prices, etc. In this paper, we present a formal approach to the clients-providers negotiation of distributed energy management. The models are described in our recently introduced REMES HDCL language, with timed automata semantics that allows us to apply UPPAAL-based tools for model-checking various scenarios of service negotiation. Our target is to compute ways of reaching the price- and reliability-optimal values of the utility function, at the end of the service negotiation.

1 Introduction

Service-oriented systems (SOS) represent a promising approach that accommodates the necessary conceptual foundation to provide quicker application development out of loosely coupled software entities, called services. The SOS paradigm also provides a way to connect new systems and services with legacy systems. Service negotiation is a key technique towards deciding and choosing the most suitable service, out of possibly many services delivering similar functionality but having different response times, resource usages, prices, etc.

The literature describes several rather theoretical results that tackle this topic [1–4] but lack constructs for formal analysis. The benefit of attaching such support to a service negotiation protocol is the capability of verifying if the negotiation design meets its specified requirements. Also, formal verification allows one to compute various quality-of-service (QoS)- optimal paths corresponding to different negotiation scenarios.

Motivated by the above, in this paper we describe the modeling and formal analysis of a distributed energy management in an open energy market, similar to one described by Mobach [5]. In an open energy market the traditional energy management does not suffice anymore, since it is required to facilitate interactions between market participants; this means that the management should be supported by a model that allows energy providers to establish agreements with energy consumers w.r.t. the supply of energy. The model of the energy market is described in Section 3.

Such a model involving customer-provider negotiation needs to be analyzed for various strategies that aim at reaching an agreement beneficial for both sides, against specified requirements. The goal of the analysis presented in this paper is also to validate our service-oriented modeling and analysis framework, that is briefly recalled in Section 2. The framework consists of the resource-aware timed behavioral modeling language REMES [6], reviewed in Section 2.1, and its underlying formal model described in terms of timed automata (TA) networks [7,8] (see Section 2.2). The negotiation model is obtained by composing REMES services, within a corresponding textual service composition language called Hierarchical Dynamic Composition Language (HDCL) (see Section 4), via operators that have been defined formally in our previous work [9]. The salient point of the approach is the fact that the obtained negotiation model can be analyzed against safety, timing, and utility constraints, for all possible behaviors of the parties. This can be achieved by transforming the negotiation model into a TA formal framework, which has a precise underlying semantics that allows its analysis with UPPAAL tools, for functional and extra-functional behaviors (timing and resource-wise behaviors) [10]. We show how to compute the price- and reliability-optimal values of the utility function, at the end of the service negotiation. The analysis of the energy negotiation process and its results are described in Section 5. Last but not least, we present some relevant related work in Section 6, before concluding the paper in Section 7.

2 Background

In this section we briefly overview the preliminaries on the REMES modeling language and the timed automata formalism, needed to comprehend the rest of the paper.

2.1 REMES - a Language for Behavioral Modeling of SOS

To describe service behavior in SOS, we use the dense-time hierarchical modeling language called REMES [6, 9]. The language is well-suited for abstract modeling, it is hierarchical, has an input/ouput distinction, a well-defined formal semantics, and tool support for SOS modeling and formal analysis [1] [10,11]. The formal analysis is accomplished by semantic transformation of REMES models into timed automata (TA) [7] or priced timed automata (PTA) [12], depending on the analysis type [10].

A service in REMES can be described graphically (as a mode), or textually, by a list of attributes (i.e., service type, capacity, time-to-serve, status, service precondition, and postcondition) exposed at the interface of the REMES service. A REMES service can be atomic, composite, but also employed in various types of compositions, resulting in new, more complex services. In order to model the synchronized behavior of parallel services we have previously introduced a special kind of REMES mode, called AND / OR mode. By the semantics of the mode, in an AND or an OR mode, the services finish their execution simultaneously, from an external observer's point of view. However, if the mode is employed as an AND mode, the subservices are entered at the same time, and their incoming edges are not constrained by any boolean enabling condition, called guard; in comparison, an OR mode assumes that one or all subservices are entered based on the guards annotated on the incoming edges. Services that belong to this type of REMES mode and that have to synchronize their behavior at the end of their execution communicate via $\|_{\mathit{SYNC\text{-}and}}$ (all services take their respective exit edges at the same time

[1] More information available at `http://www.idt.mdh.se/personal/eep/reseide/`

and mode finishes its execution), or $||_{SYNC\text{-}or}$ (the mode finishes its execution as soon as one service has taken an exit edge) operators, respectively (see our previous work [9]).

In order to manipulate services, REMES supports service creation, deletion, composition, and replacement via REMES interface operations. An example of a create service operator is given in Eq. 1. Alongside the above operations, REMES is accompanied by a hierarchical dynamic composition language (HDCL) that facilitates modeling of nested sequential, parallel or synchronized services and their compositions.

$$\begin{aligned}
\textbf{[pre]} &: service\_name == \mathsf{NULL} \\
\mathsf{create} &: Type \times N \times N \times {''passive''} \times (\Sigma \to bool) \times (\Sigma \to bool) \to service\_name \qquad (1)\\
\textbf{\{post\}} &: service\_name \neq \mathsf{NULL} \wedge Type \in \{\text{web service, network service, embedded} \wedge \\
& \wedge\, capacity \geq 0 \wedge time - to - serve \geq 0 \wedge status = {''passive''}
\end{aligned}$$

Our system is composed of REMES services that can be analyzed by transforming them into a formal network of TA that have precise semantics and can be model-checked against relevant properties (see the following section). In our recent work, we have introduced an analyzable negotiation model into the REMES language [13], that is, an analyzable high-level description of the negotiation between service clients and service providers. The model has an implicit notion of time and supports annotations in terms of price, quality, etc., all modeled by the REMES textual service composition language HDCL. The crux of the model is that it has a formal TA semantics, which lets one verify various model properties, for all possible executions. For a more thorough description of the REMES language, we refer the reader to our previous work [6,9,13,14].

2.2 Timed Automata

A timed automaton (TAn) [7,8] is a finite-state machine enriched with a set of clocks. All clocks are synchronized and assumed to be real-valued functions of time elapsed between events. In this work we use TA, as defined in the UPPAAL model-checker, which allows the use of data variables [15–17].

Let us assume a finite set of real-valued variables C ranging over x, y, etc., standing for clocks, V a finite set of all data (i.e., array, boolean, or integer), and a finite alphabet Σ ranging over a, b, etc., standing for actions. A clock constraint is a conjunctive formula of atomic constraints of the form $x \sim n$ or $x - y \sim n$ for $x, y \in C, \sim \in \{<, \leq, =, \geq, >\}$ and $n \in N$. The elements of $\mathcal{B}(C)$ are called *clock constraints* over C. Similarly, we use $\mathcal{B}(V)$ to stand for the set of *non-clock constraints* that are conjunctive formulas of $i \sim j$ or $i \sim k$, where $i, j \in V$, $k \in \mathbb{Z}$ and $\sim \in \{<, \leq, =, \neq, \geq, >\}$. We use $\mathcal{B}(C, V)$ to denote the set of formulas that are conjunctions of clock constraints and non-clock constraints.

Definition 1. *A timed automaton A is a tuple $(L, l_0, C, V, I, Act, E)$ where: L is a finite set of locations, l_0 is the initial location, C is a finite set of clocks, V is a finite set of data variables, $I : L \to \mathcal{B}(C)$ assigns (clock) invariants to locations, $Act = \Sigma \cup \{\tau\}$ is a finite set of actions, where $\tau \notin \Sigma$ denotes internal or empty actions without synchronization, $E \subseteq L \times \mathcal{B}(C, V) \times Act \times R \times L$ is the set of edges, where R denotes the (clock) reset set. In the case of $(l, g, a, r, l') \in E$, we write $l \overset{g,a,r}{\longrightarrow} l'$, where l is the source location, l' is the target location, g is a guard, a boolean condition that must hold in order for the edge to be taken, a is an action, and r is a simple clock reset.* ■

The semantics of TA is defined in terms of a labeled transition system. A state of a TAn is a pair (l, u), where l is a location, and $u : C \rightarrow R_+$ is a clock valuation. The initial state (l_0, u_0) is the starting state where all clocks are zero. There are two kinds of transitions: delay transitions and discrete transitions.

*Delay transition*s are the result of time passage and do not cause a change of location. More formally, we have $(l, u) \xrightarrow{d} (l, u \oplus d)$ if $u \oplus d' \models I(l)$ for $0 \leq d' \leq d$. The assignment $u \oplus d$ is the result obtained by incrementing all clocks of the automata with the delay d.

Discrete transitions are the result of following an enabled edge in a TAn. Consequently, the destination location is changed from the source location to the new target location, and clocks may be reset. More formally, a discrete transition $(l, u) \xrightarrow{a} (l', u')$ corresponds to taking an edge $l \overset{g,a,r}{\rightarrow} l'$ for which the guard g is satisfied by u. The clock valuation u' of the target state is obtained by modifying u according to updates r such that $u' \models I(l')$.

Reachability analysis is one of the most useful analysis to perform on a given TAn. The reachability problem can be defined as follows: Given two states of the system, is there an execution starting at one of them that reaches the other? The reachability analysis can be used to check that an error state is never reached, or just to check the sanity of the model. A network of TA, $A_1 \| ... \| A_n$, over C and Act, is defined as the parallel composition $A_1 \| ... \| A_n$ over C and Act. Semantically, a network describes a timed transition system obtained from the components, by requiring synchrony on delay transitions, and discrete transitions to synchronize on complementary actions (i.e., $a?$ (receive synchronization) is complementary to $a!$ (send synchronization)).

Properties of TA can be specified in the Timed Computation Tree Logic (TCTL), which is an extension of Computation Tree Logic (CTL) with clocks. CTL is a specification language for finite-state systems used to reason about sequence of events. Let AP be a set of atomic propositions, $p \in AP$. In this paper, a CTL formula ϕ is defined as follows:

$$\phi ::= \top \mid p \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \rightarrow \phi_2 \mid EF\phi \mid AF\phi \mid AG\phi$$

Each CTL well-defined formula is a pair of symbols. The first operator is a path operator, either A ("for All paths"), or E ("there Exists a path"). The latter operator, a temporal operator, is one of the following: F ("in a Future state"), or G ("Globally in the future"). For example $EF\phi$ means that there exists a path such that ϕ is eventually satisfied and it is called a reachability property. More details on CTL and TCTL can be found in earlier work of Alur et al. [18, 19]. In the next section we present the details of the distributed energy management case study.

3 Energy Negotiation Model in REMES HDCL

The energy management system includes an energy consumer (i.e., client) that creates a request and communicates with energy providers via a mediator. A request contains information about requested amount of energy, required price per unit of energy, and expected reliability for energy to be provided. The supply of energy is based on a negotiation carried out between consumers and providers in possibly more than one round, assuming a certain strategy. The negotiation relies on advertisements, where energy providers specify the type of energy to be sold (i.e. depending on the energy source, diesel generators, wind turbine, etc.), available amount of energy, its reliability, and price per unit of energy. In this paper's negotiation model we assume an iterative form of a Contract Net Protocol (CNP). In the CNP there exist the following roles: the client (an energy consumer), the manager (a negotiation mediator), and the contractor (an energy provider). The manager gets a request from a client and aims at finding an appropriate contractor to fulfill the request via call for proposals (CFP). Based on the response from contractors the manager decides which offers to present to a client. Depending on the implemented strategy in each round both the contractors and clients aim to improve on their previous proposals and request, in order to come closer to the consensus.

The energy consumer is assumed to have a varying energy demand that has to be satisfied over a period of time (i.e., certain periods of the day have higher energy demand than the others), while at the same time energy providers have varying energy capacity. In this model, a single day is considered (see Fig. 1), with consumer requests coming every two hours. Every two hours a new negotiation starts and should provide energy for two subsequent hours. A consumer initiates the negotiation just before the moment the energy is to be claimed and used. After a request is created, the mediator negotiates with the available energy providers, on behalf of the consumer, creating competition between energy providers. As a result of each request an agreement should be signed covering the desired energy over a defined period of time. It might be the case that involved parties do not reach a consensus and in that case no agreement is established, meaning that the client might be out of energy for that period of time.

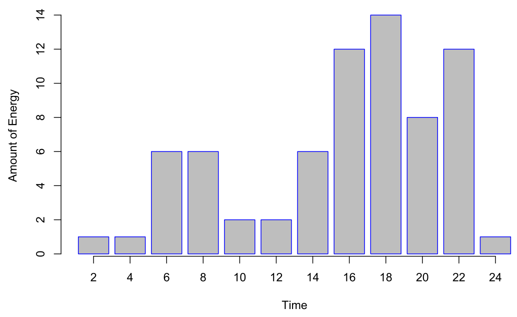


Fig. 1. An energy demand over a day

In our model, we have implemented three scenarios in which customers have encoded different behavior:

- *Scenario 1*: A customer has maximum bound on the price and the final acceptable price cannot be more than 20 price units higher than the initial requested price;
- *Scenario 2*: A customer has no maximum price value, the negotiation can continue until an agreement is conceived;
- *Scenario 3*: A customer adapts maximum price trying to get as close as possible to the offered price, but at the same time not to pay more than double initial price. The

idea behind this scenario is to get an agreement in the smallest possible number of price negotiations.

During the negotiation process the provider is not aware which strategy the client uses. In this paper, we provide a REMES - based description of the distributed energy management that is furthermore translated to the TA formal framework and analyzed against safety, timing, and utility constraints (described as a weighted sum of negotiation preferences). In the following section we will provide a REMES HDCL-based model of the energy negotiation described above.

4 REMES HDCL - Based Energy Negotiation Model

To enable a systematic and analyzable way to model the energy negotiation process, as described in Section 3, we provide the REMES HDCL description of the model. The model is based on the set of REMES interface operations and the hierarchical textual language HDCL [9].

Table 1. Service declaration

00	**declare** Service ::= <	**22**	**create** EP2 (web service, 2, 10, idle,
01	service type : {web service},	**23**	(energy_amount == ea2 $\wedge$
02	capacity : N,	**24**	min_ep2 $\leq$ ppue $\leq$ max_ep2 $\wedge$
03	time_to_serve : N,	**25**	energy_reliability == r_ep2),
04	status : { passive, idle, active},	**26**	(energy_amount == ea2-k $\wedge$
05	precondition : predicate,	**27**	min_ep2 $\leq$ ppue $\leq$ max_ep2 $\wedge$
06	postcondition : predicate >	**28**	energy_reliability == r_ep2)) : Service
07	**create** Mediator (web service, 2, 10, idle,	**29**	**declare** List ::= <[service_name$_0$: Service, . . .,
08	(req$_{\text{client}}$ == false, contract == false),	**30**	service_name$_n$: Service]>
09	(req$_{\text{client}}$ == true, contract == true)) : Service	**31**	**create** list_request : List
10	**create** Client (web service, 5, 20, idle,	**32**	**create** list_offer : List
11	(energy_amount == 0 $\wedge$ t == 0s	**33**	**add** Client list_request
12	$\wedge$ min_c $\leq$ ppue $\leq$ max_c),	**34**	**add** Manager list_request
13	(energy_amount == k $\wedge$ t $\leq$ 20s $\wedge$)	**35**	**add** EP1 list_offer
14	min_c $\leq$ ppue $\leq$ max_c) : Service	**36**	**add** EP2 list_offer
15	**create** EP1 (web service, 5, 15, idle,	**37**	**add** Manager list_offer
16	(energy_amount == ea1 $\wedge$		
17	min_ep1 $\leq$ ppue $\leq$ max_ep1 $\wedge$		
18	energy_reliability == r_ep1),		
19	(energy_amount == ea1-k $\wedge$		
20	min_ep1 $\leq$ ppue $\leq$ max_ep1$\wedge$		
21	energy_reliability == r_ep1)) : Service		

The model assumes that we first have to declare and instantiate all participating services using REMES interface operations. We model one energy consumer, two energy providers and one mediator that represents the interests of all negotiation participants as shown in Table 1 (lines 00-28). However, we have to point out that in case it would be needed to model more than one energy consumer, and more than two energy providers, the described model would be able to support it. For each negotiation participant we have provided a list of service attributes, including their pre-, and postcondtions.

Next, to model the composition of services we need to create the lists (lines 29-32 in Table 1) and add the services to the appropriate lists (see Table 1 lines 33-37). In our approach we model service negotiation as a service composition via the parallel with synchronization protocol modeled by the operator $\|_{SYNC\text{-}and}$. Services that communicate via $\|_{SYNC\text{-}and}$ operator belong to the special type of REMES mode, called AND mode. By the semantics of the AND mode, the services connected by this operator start and finish their execution simultaneously.

Finally, our model of service negotiation is defined by the following:

```
bool contract := false;
 clock h := 0;
 DCL_req ::= (list_request, $\|_{SYNC\text{-}and}$, $req_{client}$)
 DCL_offer ::= (list_offer, $\|_{SYNC\text{-}and}$, $req_{provider}$)
 DO
     p_offer := negotiation($param_p$)
     c_request := negotiation($param_c$)
 OD (c_request < p_offer) $\wedge$ h $\leq$ 24)
    contract := true;
```

Requirements req_{client} and $req_{provider}$ are predicates that include both functional and extra-functional properties of services. In our case, req_{client} defines the client's request on amount of energy, price per unit of energy (ppue), and expected energy reliability (eng_rel). On the other hand, $req_{provider}$ encompasses properties of the service that is offered by a provider. Let us assume scenario 1, as described in Section 3 and a negotiation that takes place at 18 o'clock of the day (h == 18). At this specific time req_{client} is described as: h == 18 $\wedge$ energy_amount == 14 $\wedge$ req_ppue == 15,7 $\wedge$ eng_rel == 0,8. The provider's offer for a given request is: h == 18 $\wedge$ energy_amount == 14 $\wedge$ offered_ppue == 20 $\wedge$ eng_rel == 0,8. Generally, the content of the requirement might include different negotiable parameters (denoted by $param_p$ for the provider, and $param_c$ for the client), such as price, or time at which a service should be available. As soon as the requirements are known, the negotiation can start. The provider's offer is calculated via function negotiation ($param_p$) and stored in variable p_offer similar to the approach presented by Kumpel et al. [20]. In case that the provided offer has not met the client's expectation, the request (c_request) can be updated using the same function negotiation ($param_c$) but with a different parameter (here, we abstract from the function details). The negotiation process may continue as long as the participants are interested into reaching an agreement, or in case that the negotiation model is time constrained, as long as time allows. In our case, the model is time constrained, and the negotiation will continue as long as an energy supply over a day is not satisfied. The outcome of the negotiation can either be a contract (c_ request $\geq$ p_offer) or no contract (c_ request < p_offer). In our model, the contract will be signed only if the client agrees with the offered energy amount, the price per unit of energy, and the provided energy reliability. In the example presented above (scenario 1 and energy negotiation at 18 o'clock), one can notice that the requested and offered price differ and that the negotiation is needed. If we assume the same example, then the negotiation successfully finishes with a contract signed and the final agreement of the form: h == 18 $\wedge$ energy_amount == 14 $\wedge$ final_ppue == 16 $\wedge$ eng_rel == 0,8.

The REMES language is accompanied with a tool support for constructing models as one described above in a graphical form [21]. In the following section, we show a formal analysis of the described REMES negotiation model in order to check whether the available amount of energy suffices for the client's needs and at what prices the negotiation converges. Furthermore we analyze the utility-optimal functions w.r.t. the price and the energy reliability (a weighted sum of the price and the energy reliability as the negotiation preferences).

5 Formal Analysis of the Negotiation Model

5.1 The Analysis Goals

In this paper we consider a model that supports a competition between two energy providers, available for negotiation via a mediator that acts as representative of all parties involved in the negotiation process. We model the described negotiation model using out textual composition language HDCL, and then analyze the model against several requirements, such as price, time, and reliability, in order to check whether the available energy and given prices can satisfy the client's needs. Also, it is interesting to see how much time is needed for agreement to converge.

Additionally, we calculate the value of the optimal utility function as a weighted sum of negotiation preferences w.r.t. the price and the energy reliability (modeled here by a number), and model-check the trace (a sequence of actions (delays and transitions)) that leads to such state. We calculate the value of the optimal utility function in order to find points in time when the utility function is maximized. We assume the utility function to be maximized for all participants, when the difference between their initial and their final utility values either do not exist or is insignificant.

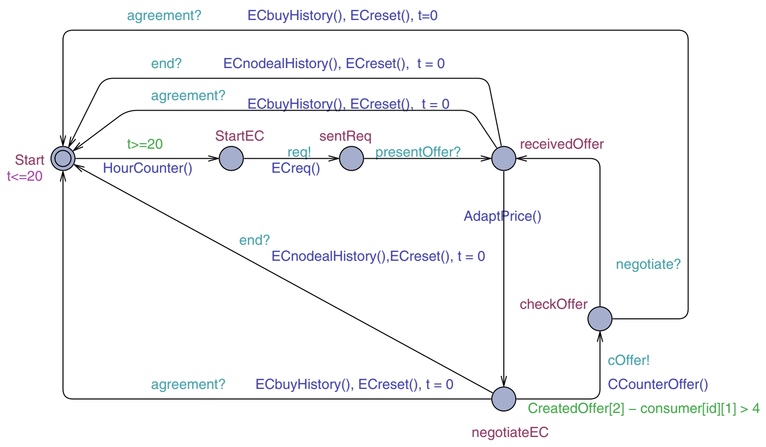
In order to ensure that our model has no deadlocks, we specify a safety UPPAAL property as follows: AG not deadlock. The given property has been verified in UPPAAL and our model satisfies it. All findings presented in the following are results of model-checking the described model in UPPAAL.

5.2 A TA Semantic Translation of the REMES Model and Analysis Results

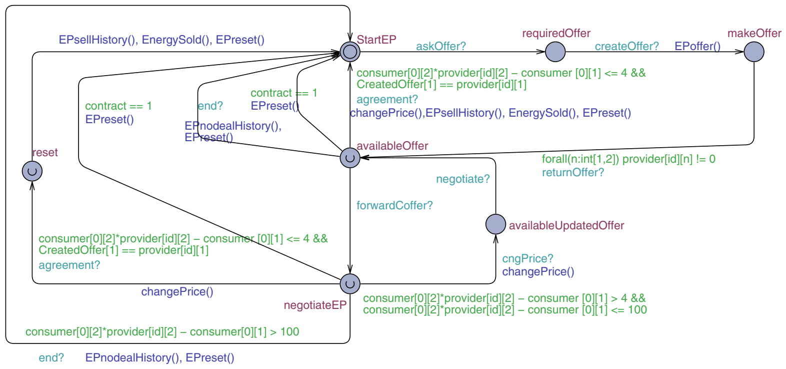
We have analyzed the REMES-based energy negotiation model, by semantically translating it into a network of TA models, in the UPPAAL [2] model-checker. The model contains five TA connected in parallel: EnergyConsumer, EnergyProvider (used to create two providers as instances of this TA), Mediator, EnergyProduction1, and EnergyProduction2. Due to space limitation, we present here the TA models of EnergyConsumer, EnergyProvider, and Mediator, shown in Fig. 2.

The TA of EnergyConsumer has six locations: Start, StartEC, sentReq, receivedOffer, negotiateEC, and checkOffer. A negotiation request is sent every 20 time units

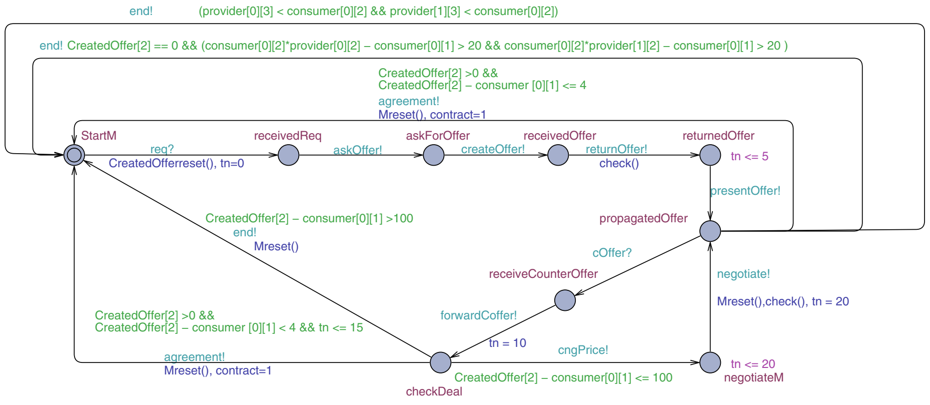
[2] See the web page `www.uppaal.org` for more information about the UPPAAL tool.


agreement?
ECbuyHistory(), ECreset(), t=0
end?
ECnodealHistory(), ECreset(), t = 0
agreement?
ECbuyHistory(), ECreset(), t = 0
Start
t<=20
t>=20
HourCounter()
StartEC
req!
ECreq()
sentReq
presentOffer?
receivedOffer
AdaptPrice()
end?
ECnodealHistory(),ECreset(), t = 0
negotiate?
checkOffer
cOffer!
CCounterOffer()
CreatedOffer[2] – consumer[id][1] > 4
agreement?
ECbuyHistory(), ECreset(), t = 0
negotiateEC

(a) A TAn of the client


EPsellHistory(), EnergySold(), EPreset()
StartEP
askOffer?
requiredOffer
createOffer?
EPoffer()
makeOffer
contract == 1
EPreset()
end?
EPnodealHistory(),
EPreset()
contract == 1
EPreset()
consumer[0][2]*provider[id][2] – consumer [0][1] <= 4 &&
CreatedOffer[1] == provider[id][1]
agreement?
changePrice(),EPsellHistory(), EnergySold(), EPreset()
reset
availableOffer
forall(n:int[1,2]) provider[id][n] != 0
returnOffer?
negotiate?
forwardCoffer?
availableUpdatedOffer
consumer[0][2]*provider[id][2] – consumer [0][1] <= 4 &&
CreatedOffer[1] == provider[id][1]
agreement?
cngPrice?
changePrice()
changePrice()
negotiateEP
consumer[0][2]*provider[id][2] – consumer [0][1] > 4 &&
consumer[0][2]*provider[id][2] – consumer [0][1] <= 100
consumer[0][2]*provider[id][2] – consumer[0][1] > 100
end?
EPnodealHistory(), EPreset()

(b) A TAn of the provider


end!
(provider[0][3] < consumer[0][2] && provider[1][3] < consumer[0][2])
end! CreatedOffer[2] == 0 && (consumer[0][2]*provider[0][2] – consumer[0][1] > 20 && consumer[0][2]*provider[1][2] – consumer[0][1] > 20)
CreatedOffer[2] >0 &&
CreatedOffer[2] – consumer [0][1] <= 4
agreement!
Mreset(), contract=1
StartM
req?
CreatedOfferreset(), tn=0
receivedReq
askOffer!
askForOffer
createOffer!
receivedOffer
returnOffer!
check()
returnedOffer
tn <= 5
presentOffer!
propagatedOffer
CreatedOffer[2] – consumer[0][1] >100
end!
Mreset()
cOffer?
receiveCounterOffer
negotiate!
Mreset(),check(), tn = 20
CreatedOffer[2] >0 &&
CreatedOffer[2] – consumer [0][1] < 4 && tn <= 15
forwardCoffer!
tn = 10
cngPrice!
agreement!
Mreset(), contract=1
CreatedOffer[2] – consumer[0][1] <= 100
checkDeal
tn <= 20
negotiateM

(c) A TAn of the negotiation mediator

Fig. 2. TA models of the negotiation participants

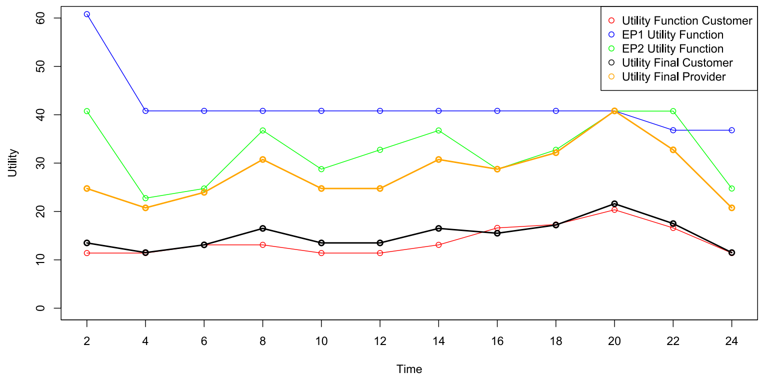


Fig. 3. Utility function change over a day for scenario 2

(t == 20), corresponding to every two hours as described in the model. Sending a request for an offer to Mediator, and receiving an offer from Mediator is modeled with channels req, and presentOffer, respectively. In case that participants need to negotiate on the current request and offer, they communicate via the broadcast channel negotiate. When an agreement is made a boolean variable contract is set to 1 and it is propagated via channel agreement, while on the other hand, in cases when no agreement has been reached (contract == 0) the channel end is used.

The TA of EnergyProvider consists of seven locations: StartEP, requiredOffer, makeOffer, availableOffer, availableUpdatedOffer, negotiateEP, and reset. A request for an offer is received from Mediator via channel askOffer. To create an offer and to further on propagate it to Mediator channels, createOffer, and returnOffer are used, respectively. In case that a request has been updated a counter offer is propagated via forwardCoffer and cngPrice broadcast channels.

The TA of Mediator has nine locations: StartM, receivedReq, askForOffer, receivedOffer, returnedOffer, propagatedOffer, receivedCounterOffer, checkDeal, and negotiateM. The automaton contains a clock variable tn, used to keep track of the time elapsed from the moment a request is received to the moment an agreement or no agreement has been signed.

In our analysis model, we encode the utility function for the consumer, and the providers, respectively, as a weighted sum of negotiation preferences (i.e., price per unit of energy and reliability given as a number and not probability), as follows:

$$\begin{aligned} utility_c &= wc_1 \times req\_ppue + wc_2 \times eng\_rel \\ utility_p &= wp_1 \times offered\_ppue + wp_2 \times eng\_rel \end{aligned} \tag{2}$$

The function is calculated for the energy consumer, both energy providers, taking into consideration the starting request/offer and the final agreement given that they have different priorities for different preferences. In case of the energy consumer reliability gets higher priority (wc_2), while in the case of the energy providers the energy price is more important (wp_1). In our case study, we consider the utility functions as described

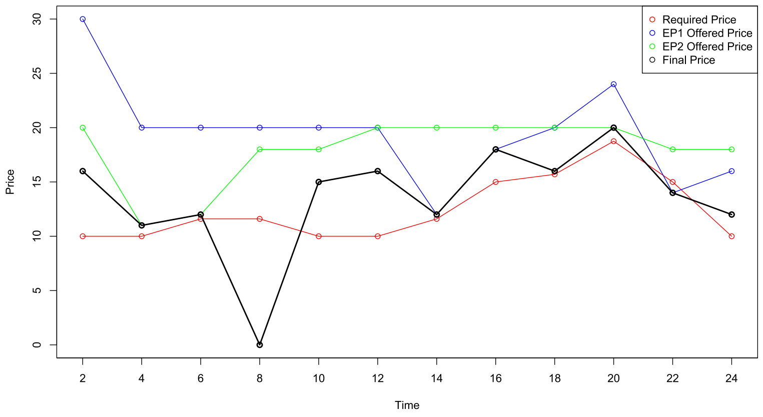


Fig. 4. Price per unit of energy for scenario 1

in Eq. 2, where wc_1, wc_2, and wp_1, wp_2 are client and provider's preferences on price and energy reliability, respectively. In the following, we present and discuss the results of the model verification in UPPAAL model-checker.

Verification shows that in scenario 1, there exists a case in which no agreement has been reached (8 o'clock), since the initially requested and offered prices were too far from each other, and since the customer had an upper bound on the price. In the same scenario, as shown in Fig. 4, in order to provide sufficient energy supply, the client is forced to spend slightly more money that initially planed, but still within the maximum price bound.

Fig. 3 depicts the utility function change over a day assuming scenario 2. Based on the history of previous request, 18 o'clock is considered as the peak hour in consumer's energy consumption. At this point in time, the utility function is maximized for each negotiation participant, respectively (the difference between initial and the final utility value either does not exist or is insignificant), meaning that the energy market favors them equally. Consequently, the consumer is prepared to request a reasonable price to make sure that he gets a required amount of energy. On the other side, the provider's offered price depends on the amount of the available energy, that is, the more energy is available the price is lower and vice versa. This means that the providers are ready for the peak hour, and have stored greater amount of energy such that they are competitive enough at the energy market. At 16 o'clock, the provider's initial and the final utility is similar, while the customer's final utility value is slightly lower than the initial value since the final price is lower than the one requested by the customer. At 20 o'clock the same situation appears, but in favor of the energy provider.

In scenario 3 we have expected that in total the client would spend more money on energy due to the fact that he was adapting his requests based on the offered prices. However, the total price is relatively close to the expected one, probably due to the mediator selecting offers on behalf of the client, which leads to the client only receiving the cheapest offers in the market, in each round. Also, the time spent to negotiate the energy supply was expected to be lower than in the other two scenarios, but it shows

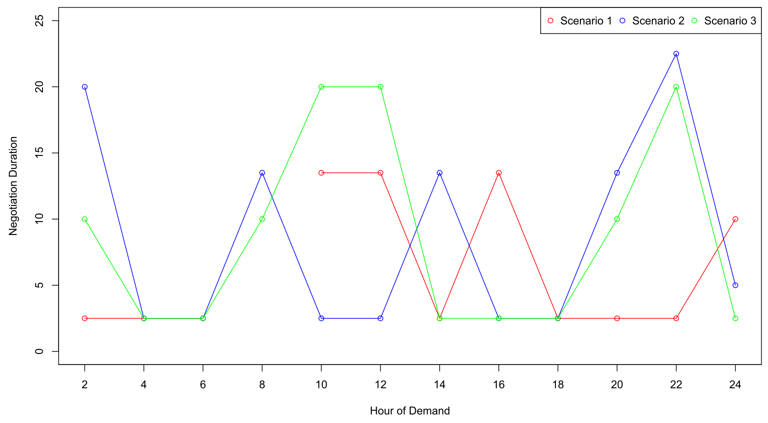


Fig. 5. Time required for negotiation

that this scenario was the most time consuming, probably due to the fact that the consumer has to adapt his acceptance threshold all over again, but still to keep within the maximum available budget (see Fig. 5). At the same time, in each negotiation round, the mediator has to check all available offers, in order to provide the client with the cheapest and most fitting one.

It was very interesting to see who owns the market in which scenario. Based on Fig. 4, in scenario 1 it is obvious that the market is own by the provider, and that even with the introduced maximum price bound, the providers were able to force the prices in their favor. Similarly, in the other two scenarios we have noticed that in scenario 3, the agreed prices are in favor of the consumer, while the final prices in scenario 2 are in favor of the energy provider. Overall, the total amount of money spent on energy in all three scenarios is very close to the initial request, with an average increase of less than 10% of the initially requested price. Before verifying the time needed in negotiation, we expected that the participants would converge toward the agreement the fastest in scenario 3. However, the results have shown the opposite, the slowest negotiation process was recorded in this scenario, possibly due to the fact that the client needed to recalculate new prices compared to the previous offers, and based on this the mediator had to ask for the new offers, always from all providers. One can notice that the least time is needed in scenario 1, while scenario 2 requires slightly more time.

6 Related Work

Mobach describes a negotiation framework based on the WS-Agreement specification [5], deployed in domains of distributed agent middleware and distributed energy management. The latter case has been simulated and evaluated through the different strategies in which energy has been distributed to the clients, including negotiation and bidding for a suitable energy source. The simulation has provided better insight in different negotiation policies, however the model lacks constructs for the formal analysis and means to provide performance analysis results. Lapadula et al. provide a description of modeling publication, discovery, negotiation, deployment, and execution of service-oriented applications in COWS [22], a language that can be translated to CMC model-checker for analysis purposes. In comparison to this approach, our framework includes analysis that caters for more than one QoS attribute (performance and reliability), while assuming time in the process too. Capodieci et al. propose an agent-based approach to model and analyze deregulated energy market [23]. In their work they adapt minority game approach that enables better distribution of the available resources. The simulation of the time flow and risk variations is done using stochastic game design. The resulting model has been simulated using JADE platform. Compared to our work the presented approach is equally fit for the modeling issues, but it lacks a possibility to exhaustively analyze the given model that could uncover more information on the issues that they describe.

7 Conclusions

In this paper, we present a case study where our recently introduced approach for automated service negotiation in REMES has been applied to model and analyze distributed energy management. The given study has been analyzed by semantically translating the REMES-based models into a network of TA to enable model-checking in the UPPAAL tool. We have focused on three scenarios as described in Section 3 by calculating the value of the optimal utility function w.r.t. the price and the energy reliability and model-checked the model to compute the traces that lead to such states. The negotiation model is time constrained, which lets one get an insight into the analysis of the time needed to reach an agreement. As future work we plan to model an auction-based energy management which would show the full potential of our negotiation model.

References

[1] Tamma, V., Wooldridge, M., Blacoe, I., Dickinson, I.: An ontology based approach to automated negotiation. In: Padget, J., Shehory, O., Parkes, D.C., Sadeh, N.M., Walsh, W.E. (eds.) AMEC 2002. LNCS (LNAI), vol. 2531, pp. 219–237. Springer, Heidelberg (2002)

[2] Resinas, M., Fernandez, P., Corchuelo, R.: A conceptual framework for automated negotiation systems. In: Corchado, E., Yin, H., Botti, V., Fyfe, C. (eds.) IDEAL 2006. LNCS, vol. 4224, pp. 1250–1258. Springer, Heidelberg (2006)

[3] Paurobally, S., Tamma, V.A.M., Wooldridge, M.: A framework for web service negotiation. TAAS 2(4) (2007)

[4] Mu-kun, C., Chi, R., Liu, Y.: Service oriented automated negotiation system architecture. In: 6th International Conference on Service Systems and Service Management, ICSSSM 2009, pp. 216–221 (2009)

[5] Mobach, D.: Agent-Based Mediated Service Negotiation. PhD thesis, Vrije University (2007)

[6] Seceleanu, C., Vulgarakis, A., Pettersson, P.: Remes: A resource model for embedded systems. In: Proc. of the 14th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 2009). IEEE Computer Society (June 2009)

[7] Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science 126(2), 183–235 (1994)

[8] Bengtsson, J., Yi, W.: Timed automata: Semantics, algorithms and tools. In: Desel, J., Reisig, W., Rozenberg, G. (eds.) ACPN 2003. LNCS, vol. 3098, pp. 87–124. Springer, Heidelberg (2004)

[9] Čaušević, A., Seceleanu, C., Pettersson, P.: Modeling and reasoning about service behaviors and their compositions. In: Margaria, T., Steffen, B. (eds.) ISoLA 2010, Part II. LNCS, vol. 6416, pp. 82–96. Springer, Heidelberg (2010)

[10] Ivanov, D., Orlic, M., Seceleanu, C., Vulgarakis, A.: Remes tool-chain - a set of integrated tools for behavioral modeling and analysis of embedded systems. In: Proceedings of the 25th IEEE/ACM International Conference on Automated Software Engineering (ASE 2010) (September 2010)

[11] Enoiu, E.P., Marinescu, R., Causevic, A., Seceleanu, C.: A design tool for service-oriented systems. In: 9th International Workshop on Formal Engineering approaches to Software Components and Architectures (FESCA 2012). ENTCS (March 2012)

[12] Alur, R., La Torre, S., Pappas, G.J.: Optimal paths in weighted timed automata. In: Di Benedetto, M.D., Sangiovanni-Vincentelli, A.L. (eds.) HSCC 2001. LNCS, vol. 2034, pp. 49–62. Springer, Heidelberg (2001)

[13] Causevic, A., Seceleanu, C., Pettersson, P.: An analyzable model of automated service negotiation. In: IEEE SOSE 2013: 7th International Symposium on Service Oriented System Engineering. IEEE (March 2013)

[14] Čaušević, A., Seceleanu, C., Pettersson, P.: Checking correctness of services modeled as priced timed automata. In: Margaria, T., Steffen, B. (eds.) ISoLA 2012, Part II. LNCS, vol. 7610, pp. 308–322. Springer, Heidelberg (2012)

[15] Bengtsson, J., Griffioen, W.D., Kristoffersen, K.J., Larsen, K.G., Larsson, F., Pettersson, P., Yi, W.: Automated verification of an audio-control protocol using uppaal. The Journal of Logic and Algebraic Programming, 163–181 (2002)

[16] Bengtsson, J., Griffioen, W., Kristoffersen, K., Larsen, K., Larsson, F., Pettersson, P., Yi, W.: Verification of an audio protocol with bus collision using UPPAAL. In: Alur, R., Henzinger, T.A. (eds.) CAV 1996. LNCS, vol. 1102, pp. 244–256. Springer, Heidelberg (1996)

[17] Bengtsson, J., Larsen, K., 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)

[18] Alur, R., Courcoubetis, C., Dill, D.: Model-checking for real-time systems. In: Proceedings of the Fifth Annual IEEE Symposium on Logic in Computer Science, LICS 1990, pp. 414–425 (June 1990)

[19] Alur, R., Courcoubetis, C., Dill, D.L.: Model-checking in dense real-time. Inf. Comput. 104(1), 2–34 (1993)

[20] Kümpel, A., Braun, I., Spillner, J., Schill, A.: (Semi-) automatic negotiation of service level agreements. In: IADIS International Conference WWW/INTERNET 2010, Timisoara, Romania, pp. 282–286 (2010)

[21] Enoiu, E.P., Marinescu, R., Causevic, A., Seceleanu, C.: A design tool for service-oriented systems. In: Proceedings the 9th International Workshop on Formal Engineering approaches to Software Components and Architectures (FESCA). Electronic Notes in Theoretical Computer Science (ENCTS), vol. 295, p. 95. Elsevier (May 2013)

[22] Lapadula, A., Pugliese, R., Tiezzi, F.: Service discovery and negotiation with cows. Electron. Notes Theor. Comput. Sci. 200, 133–154 (2008)

[23] Capodieci, N., Cabri, G., Pagani, G.A., Aiello, M.: An agent-based application to enable deregulated energy markets. In: 2012 IEEE 36th Annual Computer Software and Applications Conference, pp. 638–647 (2012)

Towards the Typing of Resource Deployment[*]

Elena Giachino and Cosimo Laneve

Dept. of Computer Science and Engineering, Università di Bologna – INRIA FOCUS
{giachino,laneve}@cs.unibo.it

Abstract. In cloud computing, *resources* as files, databases, applications, and virtual machines may either scale or move from one machine to another in response to load increases and decreases (*resource deployment*). We study a type-based technique for analysing the deployments of resources in cloud computing. In particular, we design a type system for a concurrent object-oriented language with dynamic resource creations and movements. The type of a program is *behavioural*, namely it expresses the resource deployments over periods of (logical) time. Our technique admits the inference of types and may underlie the optimisation of the costs and consumption of resources.

1 Introduction

One of the prominent features of cloud computing is elasticity, namely the property of letting (almost infinite) computing resources available on demand, thereby eliminating the need for up-front commitments by users. This elasticity may be a convenient opportunity if resources may go and shrink automatically at a fine-grain when user's needs change. However, current cloud technologies do not match this fine-grain requirement. For example, Google AppEngine automatically scales in response to load increases and decreases, but it charges clients by the cycles (type of operations) used; Amazon Web Service charges clients by the hour for the number of virtual machines used, even if a machine is idle [2].

Fine-grained resource management is an area where competition between cloud computing providers may unlock new opportunities by committing to more precise cost bounds. In turn, such cost bounds should encourage programmers to pay attention to resource managements (that is, releasing and acquiring resources only when necessary) and allow more direct measurement of operational and development inefficiencies.

In order to let *resources*, such as files or databases or applications or memories or virtual machines, be deployed in cloud machines, the languages for programming the cloud must include explicit operations for creating, deleting, and moving resources – *resource deployment operations* – and corresponding software development kits should include tools for analysing resource usages. It is worth to observe that the leveraging of resource management to the programming language might also open opportunities to implement Service Level Agreements

[*] Partly funded by the EU project FP7-610582 ENVISAGE: Engineering Virtualized Services.

(SLAs) validation via automated test infrastructure, thus offering the opportunity for third-party validation of SLAs and assessing penalties appropriately.

We study resource deployment (in cloud computing) by extending a simple concurrent object-oriented model with lightweight primitives for dynamic resource management. In our model, resources are *groups of objects* that can be dynamically created and can be moved from one (*virtual*) machine to another, called *deployment components*. We then define a technique for analysing and displaying resource loads in deployment components that is amenable to be prototyped.

The object-oriented language is overviewed in Section 2 by discussing in detail a few examples. In Section 3, we discuss the type system for analysing the resource deployments. Our technique is based on so-called *behavioural types* that abstractly describe systems' behaviours. In particular, the types we consider record the creations of resources and their movements among deployment components. They are similar to those ranging from languages for session types [7] to process contracts [17] and to calculi of processes as Milner's CCS or pi-calculus [19, 20]. In our mind, behavioural types are intended to represent a part of SLA that may be validated in a formal way and that support compositional analysis. Therefore they may play a fundamental role in the negotiation phase of cloud computing tradings.

The behavioural types presented in Section 3 are a simple model that may be displayed by highlighting the resource load of deployment components using existing tools. We examine this issue in Section 4. Related works are discussed in Section 5.

The aim of this contribution is to overview our type system for analysing resource deployments. Therefore the style is informal and problems and (our) solutions are discussed by means of examples. The details of the technique, such as the system for deriving behavioural types automatically and the correctness results, can be found in the forthcoming full paper.

2 `dcABS` in a Nutshell

Our study targets an `ABS`-like language. `ABS` [13] is a basic abstract, executable, object-oriented modelling language with a formal semantics. In this language, method invocations are asynchronous: the caller continues after the invocation and the called code runs on a different task. Tasks are cooperatively scheduled: every group of objects, called *cog*, has at most one active task at each time. Tasks running on different cogs may be evaluated in parallel, while those running on the same cog must compete for the lock and interleave their evaluation. The active task of a cog explicitly returns the control in order to let other tasks progress. Synchronisations between caller and callee is explicitly performed when callee's result is strictly necessary by using *future variables* (see [5] and the references in there).

In our language, which is called `dcABS`, programmers may define *a fixed number* of (virtual) cloud computing machines, called *deployment components* (deployment component *do not scale*), and may use a very basic management of

Table 1. A simple dcABS program

```
1  // class C declaration:
2    class C {
3      Bool m (C x) {
4        if (@this != @x) moveto @x else moveto d1;
5        return true; }
6    }
7
8  // available deployment components declaration:
9    data DCData = d0, d1, d2, d3;
10
11 //main statement:
12   C x1 = new cog C( ); moveto d1;
13   C x2 = new cog C( ); moveto d2;
14   C x3 = new cog C( ); moveto d3;
15   Fut<Bool> f1 = x1!m(x2);
16   Fut<Bool> f2 = x2!m(x3);
17   Bool b1 = f1.get;
18   Fut<Bool> f3 = x3!m(x2);
19   Bool b2 = f2.get;
20   Bool b3 = f3.get;
```

resources that enables cogs movements from one deployment component to another (*cogs represents generic resources*, such as group of computing entities, databases, virtual memories and the corresponding management processes). In `dcABS`, we also assume that *method invocations are synchronised in the same method body where they occur*, except for the main statement. This constraint largely simplifies the analysis and augment its precision because it reduces the nondeterminism.

We illustrate the main features of `dcABS` by means of examples. The details of the syntax and semantics of `dcABS` can be found in the (forthcoming) full paper. Table 1 displays a simple `dcABS` program. Programs consist of three parts: (i) a collection of class definitions, (ii) a declaration of the deployment components that are available, and (iii) a main statement to evaluate. Classes contain field and method declarations. In the above table, there is one class definition that covers lines 2–6, the deployment components are declared at line 9, and the main statement covers lines 12–20. The evaluation of the main statement is performed in the special cog `start` that is located on the deployment component that is declared first; in our example this is `d0`.

Line 12 contains a definition of `dcABS`: it creates a new object of class `C` in a new cog, locally deployed, and stores a reference to the new object in the variable `x1`. The subsequent statement `moveto d1` specifies the migration of the current cog, i.e. cog `start`, from the current deployment component `d0` to the deployment component `d1`.

Table 2. A `dcABS` recursive program

```
    class D {
      Bool move ( ) { moveto d1 ; return true ; }

      Bool multi_create (Int n) {
        if (n<=0) return true ;
        else { D x = new cog D ( ) ;
          Fut<Bool> f = x!multi_create(n-1) ;
          Bool u = f.get ;
          Fut<Bool> g = x!move( ) ;
          Bool v = g.get ;
          return true ; } }
    }
```

Lines 15, 16 and 18 display method invocations. As mentioned above, in `dcABS` invocations are *asynchronous*: the caller continues executing *in parallel with* the called method, which runs in a dedicated task within the cog where the receiver object resides. For example, line 15 corresponds to spawning the instance of the body of method `m` in a new task that is going to run in the cog of the object referred by `x1`. A *future reference* to the returned value is stored in the variable `f1` that has type `Fut<Bool>`. This means that the value is not ready yet and, when it will be produced (in the future), it will have type `Bool`. Line 17 enforces the retrieval of such value by accessing to the corresponding future reference and waiting for its availability, by means of the operation `get`. Since method invocations are asynchronous, the two invocations in lines 15 and 16 are executed concurrently. The invocation at line 15 is then synchronised at line 17, but the one at line 16 may continue concurrently with the invocation of line 18, until they are both synchronised.

The invocations in the main statement execute three instances of method `m`. Every instance verifies if the receiver object is co-located with the argument object and, in case, it performs either a deployment to let the corresponding cogs be co-located or a deployment to the component `d1`. The expression `@x` of line 4 points to the deployment component where the (cog of the) object referred by `x` resides.

Table 2 shows a class definition `D` with a recursive method `multi_create`. This method creates n new cogs co-located with the caller object and moves them to the deployment component `d1`.

Analysing the cog-deployment of the programs in Tables 1 and 2 is not straightforward. For example, significant questions regarding Table 1 are: (i) *what is the cog-load of the component* `d1` *during the lifetime of the main statement*? (ii) *Can the component* `d0` *be garbage-collected after a while in order to optimise resource usages*? Let the main statement of Table 2 be

```
// available deployment components declaration:
  data DCData = d0, d1 ;

//main statement:
   D x = new cog D() ;
   Fut<Bool> f = x!multi_create(10) ;
```

Then, an important question about Table 2 is: (iii) *is there an upper bound to the number of cogs deployed to* `d0`? The technique we study in the following sections lets us to answer to such kind of questions in a formal way.

3 Behavioural Types for Resource Deployment

Our technique for analysing resource deployments in `dcABS` programs is mostly based on our past experience in designing type inference systems for analysing deadlock-freedom of concurrent (object-oriented) languages [8–10].

A basic ingredient of every type system is the definition of the association of types with language constructs. The type system of `dcABS` associates an abstract deployment behaviour to every statement and expression. Formally, the association is defined by the typing judgment

$$\Gamma\,;n \vdash_c s \,:\, \mathbb{b} \;\triangleright\; \Gamma'\,;n' \tag{1}$$

to be read as: in an environment Γ and at a timestamp n, the statement s of an object whose cog is c has a type $\mathbb{b}$ and has effects Γ' and n'. The pair Γ' and n' is used to type the continuation. To explain (1), consider the line 12 of Table 1:

```
12   C x1 = new cog C( ); moveto d1;
```

The statement `C x1 = new cog C( );` has two effects: (i) creating a new co-located cog (with a fresh name, say c_1) , and (ii) populating this new cog with a new object whose value is stored in `x1`. As regards (i), there is a deployment of the new cog at the deployment component where the current cog c resides. We define this behaviour by means of the type

$$c_1 \mapsto c$$

As regards (ii), we record (in the typing judgment) the name of the cog of `x1`. In particular, variable assignment may propagate cog names throughout the program and this may affect the behavioural types. That is, our type system includes the *analysis of aliases* (*c.f.* Γ' in (1) is an update of Γ). In particular, in order to trace propagations of names, we associate to each variable a so-called *future record*, ranged over by $\mathbb{r}$ and defined in Table 3. A future record may be either (i) a dummy value $\_$ that models primitive types, or (ii) a record name X that represents a place-holder for a value and can be instantiated by substitutions, or (iii) $[cog{:}c, \overline{x}{:}\overline{\mathbb{r}}]$, which defines an object with its cog name c and

Table 3. Future records and behavioural types of `dcABS`

$\mathbb{r} ::= \_ \mid X \mid [cog{:}c, \overline{x}{:}\overline{\mathbb{r}}] \mid \mathit{fut}(\mathbb{r})$	future record
$\mathbb{b} ::= 0 \mid \langle c \mapsto c' \rangle^{n \div n} \mid \langle c \mapsto \mathtt{d} \rangle^{n \div n} \mid \langle \mathtt{C!m}(\overline{\mathbb{r}}) \to \mathbb{r}' \rangle^{m \div n}$ $\mid \mathbb{b} + \mathbb{b} \mid \mathbb{b} \,[]\, \mathbb{b} \mid \langle \mathbb{b} \rangle^{m \div n}$	behavioural type

the values for fields of the object, or (*iv*) $\mathit{fut}(\mathbb{r})$, which is associated to method invocations returning a value with record $\mathbb{r}$. As regards Line 12, since `C` has no field, we record in the environment Γ' of (1) the binding `x1`: $[cog : c_1]$, where c_1 is a fresh cog name.

The statement `moveto d1` corresponds to migrating the current cog (*i.e.* c) to the deployment component `d1`. This is specified by the type

$$c \mapsto \mathtt{d1}.$$

The above ones are the basic deployment informations of our type system. We next discuss the management of method invocations, which is the major difficulty in the design of the type system. In fact, the execution of methods' bodies may change deployment informations and these changes, because invocations are asynchronous, are the main source of imprecision of our analysis. Consider, for example, line 15 of Table 1

```
15    Fut<Bool> f1 = x1!m(x2);
```

and assume that the environment Γ (and Γ') in (1) binds method `m` as follows

$$\Gamma(\mathtt{C.m}) = ([cog : c], [cog : c']) \to \_$$

where

– $[cog : c]$ and $[cog : c']$ are the future records of the receiver and of the argument of the method invocation, respectively,
– $\_$ is the future record of the returned value (it is $\_$ because returned values have primitive type `Bool`).

(This association is defined during the typing of the method body – see below.) The behavioural type of the invocation `x1!m(x2)` is therefore $\mathtt{C!m}([cog : c_1], [cog : c_2]) \to \_$ where $\Gamma(\mathtt{x1}) = [cog : c_1]$ and $\Gamma(\mathtt{x2}) = [cog : c_2]$.

There is a relevant feature that is not expressed by the type $\mathtt{C!m}([cog : c_1], [cog : c_2]) \to \_$. The task corresponding to the invocation `x1!m(x2)` must be assumed to start when the invocation is evaluated and to terminate when the operation `get` on the corresponding future is performed – *cf.* line 17. During this interval, the statements of `x1!m(x2)` may interleave with those of the caller and those of the other method invocations therein – *cf.* line 16. To have a more precise analysis, we label the type of line 15 with the (logical) time interval in which it has an effect on the computation. Namely we write $\langle \mathbb{b} \rangle^{m \div n}$, where m and n are the starting and the ending interval points, respectively. Our type system increments logical timestamps in correspondence of

1. cog creations,
2. cog migrations,
3. and synchronisation points (`get` operations).

For example, the lines 15–20 of the code in Table 1 have associated timestamps

```
15  Fut<Bool> f1 = x1!m(x2); // timestamp: n
16  Fut<Bool> f2 = x2!m(x3); // timestamp: n
17  Bool b1 = f1.get; // timestamp: n
18  Fut<Bool> f3 = x3!m(x2); // timestamp: n + 1
19  Bool b2 = f2.get; // timestamp: n + 1
20  Bool b3 = f3.get; // timestamp: n + 2
```

As a consequence, the behavioural type of the above code is

$$\langle \mathtt{C!m}(\mathbb{r}_1, \mathbb{r}_2) \to \_ \rangle^{n \div n} \mathbin{[\!]} \langle \mathtt{C!m}(\mathbb{r}_2, \mathbb{r}_3) \to \_ \rangle^{n \div n+1} \mathbin{[\!]} \langle \mathtt{C!m}(\mathbb{r}_3, \mathbb{r}_2) \to \_ \rangle^{n+1 \div n+2}$$

where $\mathbb{r}_1$, $\mathbb{r}_2$ and $\mathbb{r}_3$ are the record types of the objects `x1`, `x2`, and `x3`, respectively. As we will see in Section 4, this will impact on the analysis by letting us to consider all the possible computations.

The syntax of behavioural types $\mathbb{b}$ is defined in Table 3. Apart those types that have been already discussed, $\mathbb{b} + \mathbb{b}'$ defines the abstract behaviour of conditionals, $\mathbb{b} \mathbin{[\!]} \mathbb{b}'$ corresponds to a juxtaposition of behavioural types, and $\langle \mathbb{b} \rangle^{m \div n}$ defines a behavioural type $\mathbb{b}$ to be executed in the interval $m \div n$. It is worth to notice that it is the combination of intervals that models the sequential and the parallel composition: two disjoint intervals specify two subsequent actions, while two overlapping intervals specify two (possibly) parallel actions. This complies with `dcABS` semantics where parallelism is not explicit in the syntax, but it is generated by the (asynchronous) invocations of methods.

We next discuss the association of a method behavioural type to a method declaration. To this aim, let us consider lines 3-5 of the code in Table 1:

```
3   Bool m (C x) {
4     if (@this != @x) moveto @x else moveto d1;
5     return true; }
```

The behaviour of `m` in `C` is given by $(\mathbb{r}, \mathbb{r}')\ \{\mathbb{b}_{\mathtt{m}}\} \to \_$, where $\mathbb{r}$ and $\mathbb{r}'$ are the future records of the receiver of the method and of the argument, respectively, $\mathbb{b}_{\mathtt{m}}$ is the type of the body and $\_$ is the future record of the returned boolean value. The records $\mathbb{r}$ and $\mathbb{r}'$ are formal parameters of `m`. Therefore, it is always the case that cog and record names in $\mathbb{r}$ and $\mathbb{r}'$ do occur linearly and *bind* the occurrences of names in $\mathbb{b}_{\mathtt{m}}$. It is worth to notice that cog names occurring in $\mathbb{b}_{\mathtt{m}}$ may be *not bound*. These *free names* correspond to `new cog` instructions.

In the case of m in C, its type is:

$$([cog : c], [cog : c']) \{ \langle c \mapsto c' \rangle^{1 \div 1} + \langle c \mapsto \mathtt{d1} \rangle^{1 \div 1} \} \rightarrow \_ \, .$$

The behavioural type for the the main statement of Table 1 is:

$$\begin{array}{l}
\langle c_1 \mapsto \mathit{start} \rangle^{1 \div 1} \ [] \ \langle \mathit{start} \mapsto \mathtt{d1} \rangle^{2 \div 2} \\
[] \ \langle c_2 \mapsto \mathit{start} \rangle^{3 \div 3} \ [] \ \langle \mathit{start} \mapsto \mathtt{d2} \rangle^{4 \div 4} \\
[] \ \langle c_3 \mapsto \mathit{start} \rangle^{5 \div 5} \ [] \ \langle \mathit{start} \mapsto \mathtt{d3} \rangle^{6 \div 6} \\
[] \ \langle \mathtt{C!m}([cog : c_1], [cog : c_2]) \rightarrow \_ \rangle^{7 \div 7} \\
[] \ \langle \mathtt{C!m}([cog : c_2], [cog : c_3]) \rightarrow \_ \rangle^{7 \div 8} \\
[] \ \langle \mathtt{C!m}([cog : c_3], [cog : c_2]) \rightarrow \_ \rangle^{8 \div 9} .
\end{array}$$

We conclude this section with the typing of the code in Table 2. Method move in D has type:

$$([cog : c]) \ \{ \langle c \mapsto \mathtt{d1} \rangle^{1 \div 1} \} \rightarrow \_$$

Method multi_create in D has type:

$$\begin{array}{l}
([cog : c], \_) \ \{ \\
\qquad \mathtt{0} \ + \\
\qquad\quad \langle c' \mapsto c \rangle^{1 \div 1} \ [] \ \langle \mathtt{D!multi\_create}([cog : c'], \_) \rightarrow \_ \rangle^{2 \div 2} \\
\qquad\quad [] \langle \mathtt{D!move}([cog : c']) \rightarrow \_ \rangle^{3 \div 3} \\
\qquad \} \ \rightarrow \ \_
\end{array} \tag{2}$$

We notice that the then-branch is typed with 0. In fact, it does not affect the method behaviour since it does not contain any deployment information nor method invocation.

4 Analysis of Behavioural Types

The analysis of behavioural types defined in Section 3 highlights the trend of cog numbers running in each deployment component over a period of (logical) time. More specifically, behavioural types are used to compute the abstract states of a system that record the deployment of cogs with respect to components. The component load is then obtained by projecting out the number of cogs in a state, which can be visualised by means of a standard graphic plotter program.

A primary item of this programme is the definition of the semantics of behavioural types. To this aim we use *deployment environments* Σ that map cog names to sets of deployment components. For example $[\mathit{start} \mapsto \{\mathtt{d0}\}]$ is the *initial* deployment environment. Behavioural types' semantics is defined by means of a transition system where states are triples $(\Sigma, \mathbb{b}, n)$ and transitions $(\Sigma, \mathbb{b}, n) \xrightarrow{m \div m'} (\Sigma', \mathbb{b}', n')$ are defined inductively according to the structure of $\mathbb{b}$. The basic rules of the transition relation are

$$\text{(MoveTo-c)}\quad \big(\Sigma,\ \langle c \mapsto c' \rangle^{m \div m},\ n\big) \xrightarrow{m \div m} \big(\Sigma[c \mapsto \Sigma(c')],\ \mathtt{0},\ max(m,n)\big)$$

$$\text{(MoveTo-d)}\quad \big(\Sigma,\ \langle c \mapsto \mathtt{d} \rangle^{m \div m},\ n\big) \xrightarrow{m \div m} \big(\Sigma[c \mapsto \{\mathtt{d}\}],\ \mathtt{0},\ max(m,n)\big)$$

$$\text{(Invk)}\quad \frac{\mathtt{C.m} = (\overline{\mathbb{r}})\{\mathbb{b}_{\mathtt{m}}\}\mathbb{r}' \qquad var(\mathbb{b}_{\mathtt{m}}) \setminus var(\overline{\mathbb{r}}, \mathbb{r}') = \overline{c} \qquad \overline{c'} \text{ are fresh} \qquad \mathbb{b}_{\mathtt{m}}[\overline{c'}/\overline{c}][\overline{\mathbb{s}}, \mathbb{s}'/\overline{\mathbb{r}}, \mathbb{r}'] = \mathbb{b}'}{\big(\Sigma, \langle \mathtt{C!m}(\overline{\mathbb{s}}) \to \mathbb{s}' \rangle^{m \div m'},\ n\big) \xrightarrow{m \div m'} \big(\Sigma,\ \langle \mathbb{b}' \rangle^{m \div m'},\ max(m,n)\big)}$$

The rules (MoveTo-c) and (MoveTo-d) update the deployment environment and return a null behavioural type. Rule (Invk) deals with method invocations and, apart from instantiating the formal parameters with the actual ones, it creates fresh cog names that correspond to the `new cog` operations in the method body. The inductive rules (that are omitted in this paper) lift the above transitions to structured behavioural types. In particular, let $m \div n \preceq m' \div n'$ if and only if $n < m'$ ($\preceq$ is a partial order). The rule for $\mathbb{b}_1 \,[\!]\, \cdots \,[\!]\, \mathbb{b}_k$ enables a transition $\xrightarrow{m \div n}$ provided $m \div n$ is $\preceq$-*minimal* in the set of transitions of $\mathbb{b}_1, \cdots, \mathbb{b}_k$.

In order to illustrate the operational semantics of behavioural types we discuss the transitions of the type of the main statement in Table 1:

$$\begin{array}{rcl} \mathbb{b}_0 & = & \langle c_1 \mapsto start \rangle^{1 \div 1} \ [\!]\ \langle start \mapsto \mathtt{d1} \rangle^{2 \div 2} \\ & & [\!]\ \langle c_2 \mapsto start \rangle^{3 \div 3} \ [\!]\ \langle start \mapsto \mathtt{d2} \rangle^{4 \div 4} \\ & & [\!]\ \langle c_3 \mapsto start \rangle^{5 \div 5} \ [\!]\ \langle start \mapsto \mathtt{d3} \rangle^{6 \div 6} \\ & & [\!]\ \langle \mathtt{C!m}([cog : c_1], [cog : c_2]) \to \_ \rangle^{7 \div 7} \\ & & [\!]\ \langle \mathtt{C!m}([cog : c_2], [cog : c_3]) \to \_ \rangle^{7 \div 8} \\ & & [\!]\ \langle \mathtt{C!m}([cog : c_3], [cog : c_2]) \to \_ \rangle^{8 \div 9}. \end{array}$$

Let $\Sigma_0 = [start \mapsto \mathtt{d0}]$. According to the semantics of behavioural types, we have

$$\begin{array}{l} (\Sigma_0\ ,\ \mathbb{b}_0,\ 0) \xrightarrow{1 \div 1} (\Sigma_1\ ,\ \mathbb{b}_1,\ 1) \xrightarrow{2 \div 2} (\Sigma_2\ ,\ \mathbb{b}_2,\ 2) \xrightarrow{3 \div 3} (\Sigma_3\ ,\ \mathbb{b}_3,\ 3) \xrightarrow{4 \div 4} (\Sigma_4\ ,\ \mathbb{b}_4,\ 4) \\ \qquad \xrightarrow{5 \div 5} (\Sigma_5\ ,\ \mathbb{b}_5,\ 5) \xrightarrow{6 \div 6} (\Sigma_6\ ,\ \mathbb{b}_6,\ 6) \end{array}$$

where, at each step $1 \le i \le 6$, the type that is evaluated is the one with interval $i \div i$, the deployment environment Σ_6 is $[start \mapsto \{\mathtt{d3}\}, c_1 \mapsto \{\mathtt{d0}\}, c_2 \mapsto \{\mathtt{d1}\}, c_3 \mapsto \{\mathtt{d2}\}]$, and the type $\mathbb{b}_6$ is $\langle \mathtt{C!m}([cog : c_1], [cog : c_2]) \to \_ \rangle^{7 \div 7} \ [\!]\ \langle \mathtt{C!m}([cog : c_2], [cog : c_3]) \to \_ \rangle^{7 \div 8} \ [\!]\ \langle \mathtt{C!m}([cog : c_3], [cog : c_2]) \to \_ \rangle^{8 \div 9}$.

In Figure 1 we have drawn the computations starting at $\big(\Sigma_6,\ \mathbb{b}_6,\ 6\big)$. Here we discuss the rightmost computation. In $\big(\Sigma_6,\ \mathbb{b}_6,\ 6\big)$, the two transitions that are possible are the method invocations with intervals $7 \div 7$ and $7 \div 8$. We perform the one with interval $7 \div 8$ and, by rule (Invk), we get $\big(\Sigma_6,\ \mathbb{b}_8,\ 7\big)$, where $\mathbb{b}_8 = \langle \mathtt{C!m}([cog : c_1], [cog : c_2]) \to \_ \rangle^{7 \div 7} \ [\!]\ \langle \langle c_2 \mapsto c_3 \rangle^{1 \div 1} + \langle c_2 \mapsto \mathtt{d1} \rangle^{1 \div 1} \rangle^{7 \div 8} \ [\!]\ \langle \mathtt{C!m}([cog : c_3], [cog : c_2]) \to \_ \rangle^{8 \div 9}$.

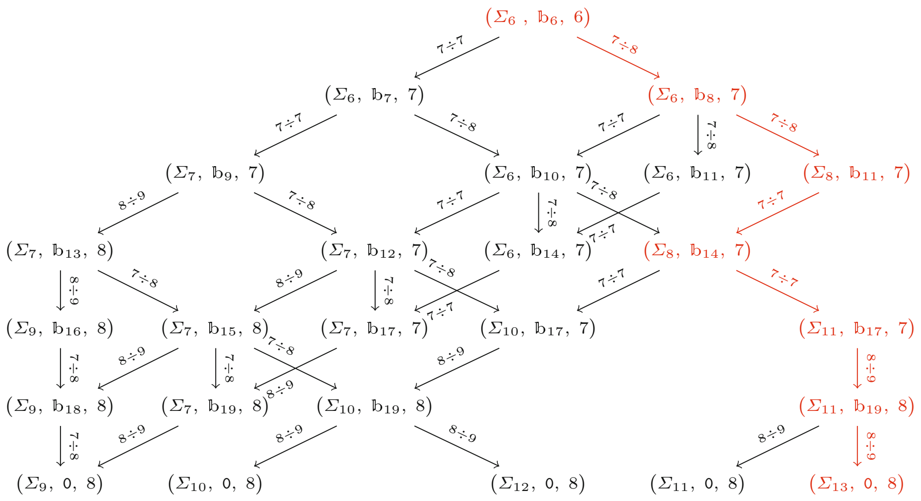


Fig. 1. An example of transition system of behavioural types

At this point there are three options: the method invocation with interval $7 \div 7$ or the evaluation of either $\langle c_2 \mapsto c_3 \rangle^{1 \div 1}$ or $\langle c_2 \mapsto \mathtt{d1} \rangle^{1 \div 1}$, both with interval $7 \div 8$ because underneath a $\langle \cdot \rangle^{7 \div 8}$ context.

By evaluating $\langle c_2 \mapsto c_3 \rangle^{1 \div 1}$, one obtains $(\Sigma_8,\ \mathbb{b}_{11},\ 7)$, where Σ_8 is $[\mathit{start} \mapsto \{\mathtt{d3}\},\ c_1 \mapsto \{\mathtt{d0}\},\ c_2 \mapsto \{\mathtt{d2}\},\ c_3 \mapsto \{\mathtt{d2}\}]$. and $\mathbb{b}_{11}$ is $\langle \mathtt{C!m}([\mathit{cog} : c_1], [\mathit{cog} : c_2]) \to \_ \rangle^{6 \div 6} \mathbin{[]} \langle \mathtt{C!m}([\mathit{cog} : c_3], [\mathit{cog} : c_2]) \to \_ \rangle^{7 \div 8}$.

In $(\Sigma_8,\ \mathbb{b}_{11},\ 7)$ only one transition is possible: the method invocation with interval $7 \div 7$. Therefore one has $(\Sigma_8,\ \mathbb{b}_{14},\ 7)$, where $\mathbb{b}_{14} = \langle \langle c_1 \mapsto c_2 \rangle^{1 \div 1} + \langle c_1 \mapsto \mathtt{d1} \rangle^{1 \div 1} \rangle^{7 \div 7} \mathbin{[]} \langle \mathtt{C!m}([\mathit{cog} : \mathtt{c}_3], [\mathit{cog} : \mathtt{c}_2]) \to \_ \rangle^{8 \div 9}$.

In the state $(\Sigma_8,\ \mathbb{b}_{14},\ 7)$ the possible transitions are those of the type $\langle \langle c_1 \mapsto c_2 \rangle^{1 \div 1} + \langle c_1 \mapsto \mathtt{d1} \rangle^{1 \div 1} \rangle^{7 \div 7}$. By letting $\langle c_1 \mapsto \mathtt{d1} \rangle^{1 \div 1}$ move, one has $(\Sigma_{11},\ \mathbb{b}_{17},\ 7)$, where $\Sigma_{11} = [\mathit{start} \mapsto \{\mathtt{d3}\},\ c_1 \mapsto \{\mathtt{d1}\},\ c_2 \mapsto \{\mathtt{d2}\},\ c_3 \mapsto \{\mathtt{d2}\}]$ and $\mathbb{b}_{17} = \langle \mathtt{C!m}([\mathit{cog} : c_3], [\mathit{cog} : c_2]) \to \_ \rangle^{8 \div 9}$. Finally, by performing two transitions labelled $8 \div 9$, one first unfolds the method invocation and then evaluates the corresponding body $\langle \langle c_3 \mapsto c_2 \rangle^{1 \div 1} + \langle c_3 \mapsto \mathtt{d1} \rangle^{1 \div 1} \rangle^{8 \div 9}$. By letting $\langle c_3 \mapsto \mathtt{d1} \rangle^{1 \div 1}$ move, the computation terminates with a deployment environment $\Sigma_{13} = [\mathit{start} \mapsto \{\mathtt{d3}\},\ c_1 \mapsto \{\mathtt{d1}\},\ c_2 \mapsto \{\mathtt{d2}\},\ c_3 \mapsto \{\mathtt{d1}\}]$.

Given a transition system $\mathcal{T}$ as the one illustrated in Figure 1, it is possible to compute the *abstract trace*, *i.e.* the sequence $\Sigma(0) \cdot \Sigma(1) \cdot \Sigma(2) \cdots$ where $\Sigma(i)$ is the deployment environment

$$\Sigma(i) : c \mapsto \cup \{ \Sigma(c) \quad | \quad \text{there is } \mathbb{b} \text{ such that } (\Sigma,\ \mathbb{b},\ i) \in \mathcal{T} \} \ .$$

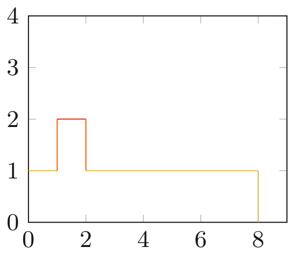
For example, letting the deployment environments of Figure 1 be

$$\begin{aligned}
\Sigma_6 &= [\mathit{start} \mapsto \{\mathtt{d3}\},\, c_1 \mapsto \{\mathtt{d0}\},\, c_2 \mapsto \{\mathtt{d1}\},\, c_3 \mapsto \{\mathtt{d2}\}]\\
\Sigma_7 &= [\mathit{start} \mapsto \{\mathtt{d3}\},\, c_1 \mapsto \{\mathtt{d1}\},\, c_2 \mapsto \{\mathtt{d1}\},\, c_3 \mapsto \{\mathtt{d2}\}]\\
\Sigma_8 &= [\mathit{start} \mapsto \{\mathtt{d3}\},\, c_1 \mapsto \{\mathtt{d0}\},\, c_2 \mapsto \{\mathtt{d2}\},\, c_3 \mapsto \{\mathtt{d2}\}]\\
\Sigma_9 &= [\mathit{start} \mapsto \{\mathtt{d3}\},\, c_1 \mapsto \{\mathtt{d1}\},\, c_2 \mapsto \{\mathtt{d1}\},\, c_3 \mapsto \{\mathtt{d1}\}]\\
\Sigma_{10} &= [\mathit{start} \mapsto \{\mathtt{d3}\},\, c_1 \mapsto \{\mathtt{d1}\},\, c_2 \mapsto \{\mathtt{d2}\},\, c_3 \mapsto \{\mathtt{d2}\}]\\
\Sigma_{11} &= [\mathit{start} \mapsto \{\mathtt{d3}\},\, c_1 \mapsto \{\mathtt{d2}\},\, c_2 \mapsto \{\mathtt{d2}\},\, c_3 \mapsto \{\mathtt{d2}\}]\\
\Sigma_{12} &= [\mathit{start} \mapsto \{\mathtt{d3}\},\, c_1 \mapsto \{\mathtt{d1}\},\, c_2 \mapsto \{\mathtt{d2}\},\, c_3 \mapsto \{\mathtt{d1}\}]\\
\Sigma_{13} &= [\mathit{start} \mapsto \{\mathtt{d3}\},\, c_1 \mapsto \{\mathtt{d2}\},\, c_2 \mapsto \{\mathtt{d2}\},\, c_3 \mapsto \{\mathtt{d1}\}]
\end{aligned}$$

we can compute the cog trend for each deployment component. Let $\Sigma(i)|_{\mathtt{d}} \overset{def}{=} \{c \mid \mathtt{d} \in \Sigma(i)(c)\}$. Then

$\Sigma(i)\|_{\mathtt{d}}$	d0	d1	d2	d3
$\Sigma(0)$	*start*	$\varnothing$	$\varnothing$	$\varnothing$
$\Sigma(1)$	c_1, *start*	$\varnothing$	$\varnothing$	$\varnothing$
$\Sigma(2)$	c_1	*start*	$\varnothing$	$\varnothing$
$\Sigma(3)$	c_1	c_2, *start*	$\varnothing$	$\varnothing$
$\Sigma(4)$	c_1	c_2	*start*	$\varnothing$
$\Sigma(5)$	c_1	c_2	c_3, *start*	$\varnothing$
$\Sigma(6)$	c_1	c_2	c_3	*start*
$\Sigma(7)$	c_1	c_1, c_2	c_1, c_2, c_3	*start*
$\Sigma(8)$	$\varnothing$	c_1, c_2, c_3	c_1, c_2, c_3	*start*

Graphically (note that `d0` starts at level 1, since at the beginning it contains the "*start*" cog):



We conclude our overview by discussing the issue of recursive invocation. To this aim, consider the type (2) of the method `multi_create` in Table 2 and the main statement

```
D x = new cog D( ); Fut<Bool> f = x!multi_create(4); Bool b = f.get;
```

whose type is:

$$\mathbb{b}_0^r = \langle c_1^r \mapsto \mathit{start} \rangle^{1 \div 1} \, [] \, \langle \mathtt{D!multi\_create}([\mathit{cog} : c_1^r], \_) \to \_ \rangle^{2 \div 2}$$

Being `d0` and `d1` the two declared deployment components, we obtain the following computation:

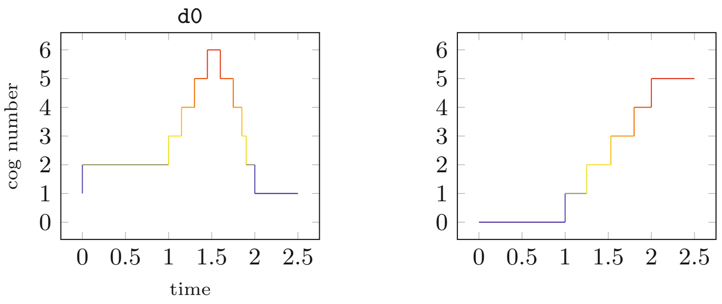
$$
\begin{aligned}
&\big([\mathit{start} \mapsto \{\texttt{d0}\}]\ ,\ \mathbb{b}^r_0,\ 0\big)\\
&\quad \xrightarrow{1\div 1} \big([\mathit{start} \mapsto \{\texttt{d0}\}, c^r_1 \mapsto \{\texttt{d0}\}]\ ,\ \mathbb{b}^r_1,\ 1\big)\\
&\quad \xrightarrow{2\div 2}\ \xrightarrow{2\div 2} \big([\mathit{start} \mapsto \{\texttt{d0}\}, c^r_1 \mapsto \{\texttt{d0}\}, c^r_2 \mapsto \{\texttt{d0}\}]\ ,\ \mathbb{b}^r_3,\ 2\big)\\
&\quad \xrightarrow{2\div 2} \cdots \xrightarrow{2\div 2} \big([\mathit{start} \mapsto \{\texttt{d0}\}, c^r_1 \mapsto \{\texttt{d0}\}, c^r_2 \mapsto \{\texttt{d0}\}, \cdots, c^r_{n-2} \mapsto \{\texttt{d0}\}]\ ,\ \mathbb{b}^r_n,\ 2\big)\\
&\quad \xrightarrow{2\div 2}\ \xrightarrow{2\div 2} \big([\mathit{start} \mapsto \{\texttt{d0}\}, c^r_1 \mapsto \{\texttt{d0}\}, c^r_2 \mapsto \{\texttt{d0}\}, \cdots, c^r_{n-2} \mapsto \{\texttt{d1}\}]\ ,\ \mathbb{b}^r_{n+2},\ 2\big)\\
&\quad \xrightarrow{2\div 2} \cdots \xrightarrow{2\div 2} \big([\mathit{start} \mapsto \{\texttt{d0}\}, c^r_1 \mapsto \{\texttt{d0}\}, c^r_2 \mapsto \{\texttt{d1}\}, \cdots, c^r_{n-2} \mapsto \{\texttt{d1}\}]\ ,\ \texttt{0},\ 2\big)
\end{aligned}
$$

where

$$
\begin{aligned}
\mathbb{b}^r_1 &= \langle \texttt{D!multi\_create}([\mathit{cog} : c^r_1], \_) \to \_\rangle^{2\div 2}\\
\mathbb{b}^r_2 &= \langle \quad \langle c^r_2 \mapsto c^r_1\rangle^{1\div 1} \ [\!]\ \langle \texttt{D!multi\_create}([\mathit{cog} : c^r_2], \_) \to \_\rangle^{2\div 2}\\
&\qquad\qquad [\!]\langle \texttt{D!move}([\mathit{cog} : c^r_2]) \to \_\rangle^{3\div 3} \quad \rangle^{2\div 2}\\
\mathbb{b}^r_3 &= \langle\ \langle \texttt{D!multi\_create}([\mathit{cog} : c^r_2], \_) \to \_\rangle^{2\div 2}\ [\!]\ \langle \texttt{D!move}([\mathit{cog} : c^r_2]) \to \_\rangle^{3\div 3}\\
&\quad \rangle^{2\div 2}\\
\ldots\\
\mathbb{b}^r_n &= \langle\ \langle\langle \cdots \langle\langle \texttt{D!move}([\mathit{cog} : c^r_{(n-2)}]) \to \_\rangle^{3\div 3}\rangle^{2\div 2}\\
&\qquad [\!]\langle\langle \texttt{D!move}([\mathit{cog} : c^r_{(n-3)}]) \to \_\rangle^{3\div 3} \cdots \rangle^{2\div 2}\\
&\qquad [\!]\langle \texttt{D!move}([\mathit{cog} : c^r_3]) \to \_\rangle^{3\div 3}\rangle^{2\div 2}\ [\!]\ \langle \texttt{D!move}([\mathit{cog} : c^r_2]) \to \_\rangle^{3\div 3}\\
&\quad \rangle^{2\div 2}\\
\mathbb{b}^r_{n+2} &= \langle\ \langle\langle \cdots \langle \texttt{D!move}([\mathit{cog} : c^r_{(n-3)}]) \to \_\rangle^{3\div 3} \cdots \rangle^{2\div 2}\\
&\qquad [\!]\langle \texttt{D!move}([\mathit{cog} : c^r_3]) \to \_\rangle^{3\div 3}\rangle^{2\div 2}\ [\!]\ \langle \texttt{D!move}([\mathit{cog} : c^r_2]) \to \_\rangle^{3\div 3}\\
&\quad \rangle^{2\div 2}
\end{aligned}
$$

We observe the following two facts: first, every transition, except the initial one, is at logical timestamp 2, because only the outermost interval is observable, while the nested intervals are only relevant to specify the order of events at the same level of nesting; second, in case of recursion the specified behaviour is potentially infinite and parameterised by the number n of transitions, which depends on the number of recursive invocations.

In visualising the results of the analysis, these two aspects pose some questions: the first one may lead us to flatten all the events at timestamp 2 as they happened in parallel, while if observing carefully the computation we notice the events follow a strict sequence; the second one may make it difficult to graphically represent the unbounded behaviour. To address the first point, we don't simply rely on the label of the transition to recognise the state of the computation, but at each interval the visualiser performs a sort of *zoom in*, so to magnify the nested behaviour. The result is the sequentialised behaviour depicted below. To address the second one, we just approximate the behaviour by letting *at most* n nested recursive invocations. The corresponding graphs are as follows, fixing $n = 8$, (note that `d0` starts at level 1, since at the beginning it contains the "*start*" cog):



In this case, the recursive behaviour corresponds to a pick of deployed cogs in the interval $2 \div 3$. This pick grows according to the value of n. The interesting property we may grasp from the graphs for `d0` is that, the upward pick in the interval $1 \div 2$ corresponds to a downward pick in the same interval of the same length. This is due to the property that, for each increment in that interval, there is a decrement, thus leaving unchanged the number of cogs in `d0` (which is 2). A different behaviour is manifested by the graph of the component `d1`. In this case, there is a growing increment of deployed cogs according to the increasing of n. The rightmost function lets us derive that the deployment component `d1` may become critical as the computation progresses.

5 Related Work

Resource analysis has been extensively studied in the literature and several methods have been proposed, ranging from static analyses (data-flow analysis and type systems) to model checking. We discuss in this section a number of related techniques and the differences with the one proposed in this paper.

A well-known technique is the so-called *resource-aware programming* [21] that allows users to monitor the resources consumed by their programs and to express policies for the management of such resources in the programs. Resource-aware programming is also available for mainstream languages, such as Java [4]. Our typing system may integrate resource-aware programming by providing static-time feedbacks about the correctness of the management, such as full-coverage of cases, correctness of the policies, etc.

Other techniques address resource management in embedded systems and mostly use performance analysis on models that are either process algebra [18], or Petri Nets [23], or various types of automata [24]. It is also worth to remind that similar techniques have been defined for web services and business processes [6, 22]. Usually, all these approaches are *invasive* because they oblige programmers to declare the cost of transitions in terms of time or in terms of a single resource. On the contrary, our technique does not assign any commitment to programmers, which may be completely unaware of resources and their management.

In [1] a quantified analysis targets `ABS` programs and returns informations about the different kinds of nodes that compose the system, how many instances of each kind exist, and node interactions. A resource analysis infers upper bounds to the number of concrete instances that the nodes and arcs represent. (The analysis in [1] does not explicitly support deployment components and cog migration; however we believe that this integration is possible.) An important difference of this analysis with respect to our contribution is that our behavioural types are intended to represent a part of SLA that may be validated in a formal way and that support compositional analysis. It is not clear if these correspondence with SLA is also possible for the models of [1].

A type inference technique for resource analysis has been developed in [11,12]. They study the problem of worst-case heap usage in functional and (sequential) object-oriented languages and their tool returns functions on the size of inputs of every method that highlight the heap consumption. On the contrary, our technique returns upper bounds *disregarding* input sizes. However, we think it is possible to extend our types to enable a transition system model that support the expressivity of [12] (our current analysis of behavioural types is preliminary and must be considered as a proof-of-concept). In these regards, we plan to explore the adoption of behavioural types that depends [3] on the input data of conditions in if-statements. We observe, anyway, that the generalisation of the results in [11, 12] to a concurrent setting has not been investigated.

Kobayashi, Suenaga and Wischik develop a technique that is very close to the one in this paper [16]. In particular, they extend pi-calculus with primitives for creating and using resources and verify whether a program conforms with resource usage declarations (that may be also automatically inferred). A difference between their technique and the one in this paper is that here the resource analysis is performed *ex-post* by resorting to abstract transition systems of behavioural types, while in [16] the analysis is done during the type checking(/inference). As discussed in [9], our technique is in principle more powerful than those verifying resource usage during the checking/inference of types.

6 Conclusions

This work is a preliminary theoretical study about the analysis of resource deployments by means of type systems. Our types are lightweight abstract descriptions of behaviours that retain resource informations and admit type inference.

The analysis of behavioural types that has been discussed in Section 4 is very preliminary. In fact, in Example 2, the resource analysis depends on the input value of the method `multi_create`. In these cases, a reasonable output of the analysis is a formula that defines the cog-load of deployment components according to the actual value in input. As discussed in Section 5, we intend to investigate more convenient behavioural type analyses, possibly by using more expressive types, such as dependent ones [3].

One obvious research direction is to apply our technique for defining an inference system for resource deployment in programming languages, such as `ABS` or `core ABS`, and prototyping it with a tool for displaying the load of deployment components. The programme is similar to the one developed for deadlock analysis [10]. The next step is then the experiment of the prototype on real programs in order to have assessments about its performance and precision.

We also intend to study the range of application of type system techniques when resources are either cloud virtual machines, or CPU, or memory, or bandwidth. The intent is to replace/complement the simulation techniques used in [14,15] with static analysis techniques based on types.

References

[1] Albert, E., Correas, J., Puebla, G., Román-Díez, G.: Quantified abstractions of distributed systems. In: Johnsen, E.B., Petre, L. (eds.) IFM 2013. LNCS, vol. 7940, pp. 285–300. Springer, Heidelberg (2013)

[2] Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R.H., Konwinski, A., Lee, G., Patterson, D.A., Rabkin, A., Stoica, I., Zaharia, M.: A view of cloud computing. Commun. ACM 53(4), 50–58 (2010)

[3] Bove, A., Dybjer, P.: Dependent types at work. In: Bove, A., Barbosa, L.S., Pardo, A., Pinto, J.S. (eds.) LerNet ALFA Summer School 2008. LNCS, vol. 5520, pp. 57–99. Springer, Heidelberg (2009)

[4] Czajkowski, G., von Eicken, T.: JRes: A resource accounting interface for Java. In: Proceedings of OOPSLA, pp. 21–35 (1998)

[5] de Boer, F.S., Clarke, D., Johnsen, E.B.: A complete guide to the future. In: De Nicola, R. (ed.) ESOP 2007. LNCS, vol. 4421, pp. 316–330. Springer, Heidelberg (2007)

[6] Foster, H., Emmerich, W., Kramer, J., Magee, J., Rosenblum, D.S., Uchitel, S.: Model checking service compositions under resource constraints. In: Proc. 6th of the European Software Engineering Conf. and the Symposium on Foundations of Software Engineering, pp. 225–234. ACM (2007)

[7] Gay, S., Hole, M.: Subtyping for session types in the π-calculus. Acta Informatica 42(2-3), 191–225 (2005)

[8] Giachino, E., Grazia, C.A., Laneve, C., Lienhardt, M., Wong, P.Y.H.: Deadlock analysis of concurrent objects: Theory and practice. In: Johnsen, E.B., Petre, L. (eds.) IFM 2013. LNCS, vol. 7940, pp. 394–411. Springer, Heidelberg (2013)

[9] Giachino, E., Kobayashi, N., Laneve, C.: Deadlock analysis of unbounded process networks. In: Baldan, P., Gorla, D. (eds.) CONCUR 2014. LNCS, vol. 8704, pp. 63–77. Springer, Heidelberg (2014)

[10] Giachino, E., Laneve, C., Lienhardt, M.: A Framework for Deadlock Detection in ABS. Software and Systems Modeling (to appear, 2014)

[11] Hoffmann, J., Aehlig, K., Hofmann, M.: Multivariate amortized resource analysis. ACM Trans. Program. Lang. Syst. 34(3), 14 (2012)

[12] Hofmann, M., Rodriguez, D.: Automatic type inference for amortised heap-space analysis. In: Felleisen, M., Gardner, P. (eds.) ESOP 2013. LNCS, vol. 7792, pp. 593–613. Springer, Heidelberg (2013)

[13] Johnsen, E.B., Hähnle, R., Schäfer, J., Schlatte, R., Steffen, M.: ABS: A core language for abstract behavioral specification. In: Aichernig, B.K., de Boer, F.S., Bonsangue, M.M. (eds.) FMCO 2010. LNCS, vol. 6957, pp. 142–164. Springer, Heidelberg (2011)

[14] Johnsen, E.B., Owe, O., Schlatte, R., Tapia Tarifa, S.L.: Dynamic resource reallocation between deployment components. In: Dong, J.S., Zhu, H. (eds.) ICFEM 2010. LNCS, vol. 6447, pp. 646–661. Springer, Heidelberg (2010)

[15] Johnsen, E.B., Owe, O., Schlatte, R., Tapia Tarifa, S.L.: Validating timed models of deployment components with parametric concurrency. In: Beckert, B., Marché, C. (eds.) FoVeOOS 2010. LNCS, vol. 6528, pp. 46–60. Springer, Heidelberg (2011)

[16] Kobayashi, N., Suenaga, K., Wischik, L.: Resource usage analysis for the pi-calculus. Logical Methods in Computer Science 2(3) (2006)
[17] Laneve, C., Padovani, L.: The *must* preorder revisited. In: Caires, L., Vasconcelos, V.T. (eds.) CONCUR 2007. LNCS, vol. 4703, pp. 212–225. Springer, Heidelberg (2007)
[18] Lüttgen, G., Vogler, W.: Bisimulation on speed: A unified approach. Theoretical Computer Science 360(1-3), 209–227 (2006)
[19] Milner, R.: A Calculus of Communicating Systems. Springer (1982)
[20] Milner, R., Parrow, J., Walker, D.: A calculus of mobile processes, ii. Inf. and Comput. 100, 41–77 (1992)
[21] Moreau, L., Queinnec, C.: Resource aware programming. ACM Trans. Program. Lang. Syst. 27(3), 441–476 (2005)
[22] Netjes, M., van der Aalst, W.M., Reijers, H.A.: Analysis of resource-constrained processes with Colored Petri Nets. In: Proceedings of the Sixth Workshop on the Practical Use of Coloured Petri Nets and CPN Tools (CPN 2005). DAIMI, vol. 576. University of Aarhus (2005)
[23] Sgroi, M., Lavagno, L., Watanabe, Y., Sangiovanni-Vincentelli, A.: Synthesis of embedded software using free-choice Petri nets. In: Proc. 36th ACM/IEEE Design Automation Conference (DAC 1999), pp. 805–810. ACM (1999)
[24] Vulgarakis, A., Seceleanu, C.C.: Embedded systems resources: Views on modeling and analysis. In: Proc. 32nd IEEE Intl. Computer Software and Applications Conference (COMPSAC 2008), pp. 1321–1328. IEEE Computer Society (2008)

Static Inference of Transmission Data Sizes in Distributed Systems

Elvira Albert[1], Jesús Correas[1],
Enrique Martin-Martin[1], and Guillermo Román-Díez[2]

[1] DSIC, Complutense University of Madrid, Spain
[2] DLSIIS, Technical University of Madrid, Spain

Abstract. We present a static analysis to infer the amount of data that a distributed system may transmit. The different locations of a distributed system communicate and coordinate their actions by posting tasks among them. A task is posted by building a message with the task name and the data on which such task has to be executed. When the task completes, the result can be retrieved by means of another message from which the result of the computation can be obtained. Thus, the transmission data size of a distributed system mainly depends on the amount of messages posted among the locations of the system, and the sizes of the data transferred in the messages. Our static analysis has two main parts: (1) we over-approximate the sizes of the data at the program points where tasks are spawned and where the results are received, and (2) we over-approximate the total number of messages. Knowledge of the transmission data sizes is essential, among other things, to predict the bandwidth required to achieve a certain response time, or conversely, to estimate the response time for a given bandwidth. A prototype implementation in the SACO system demonstrates the accuracy and feasibility of the proposed analysis.

1 Introduction

Distributed systems are increasingly used in industrial processes and products, such as manufacturing plants, aircraft and vehicles. For example, many control systems are decentralized using a distributed architecture with different processing locations interconnected through buses or networks. The software in these systems typically consists of concurrent tasks which are statically allocated to specific locations for processing, and which exchange messages with other tasks at the same or at other locations to perform a collaborative work. A decentralized approach is often superior to traditional centralized control systems in performance, capability and robustness. Systems such as control systems are often critical: they have strict requirements with respect to timing, performance, and stability. A failure to meet these requirements may have catastrophic consequences. To verify that a given system is able to provide the required quality of control, an essential aspect is to accurately predict the communication traffic among its distributed components, i.e., the amount of data to be transmitted along any execution of the distributed system.

In order to estimate the transmission data sizes, we need to keep track of the amount of data transmitted in two ways: (1) by posting asynchronous tasks among the locations, this requires building a message in which the name of the task to execute and the data on which it executes are included; (2) by retrieving the results of executing the tasks, in our setting, we use future variables [8] to synchronize with the completion of a task and retrieve the result. This paper presents a static analysis to infer a safe over-approximation of the transmission data sizes required by both sources of communications in a distributed system. Our method infers three different pieces of information:

1. *Inference of distributed locations.* As locations can be dynamically created, in a first step, we need to find out the locations that compose the system and give them abstract names which will allow us to track communications among them during the analysis. This is formalized by means of points-to analysis [14,13], a typical analysis in pointer-based languages which infers the memory locations that a reference variable can point to. In our case, locations are referenced from reference variables, thus the use of points-to analysis.
2. *Inference of number of tasks spawned.* The second step is to infer an upper bound on the number of tasks spawned between each pair of distributed locations. This is a problem which can be solved by a generic cost analysis framework such as [3]. In particular, we need to use a *symbolic* cost model which allows us to annotate the caller and callee locations when a task is spawned in the program. In essence, if we find an instruction a!m(x) which spawns a task m at location a, the cost model symbolically counts $c(this, a, m) * 1$, i.e., it counts that 1 task executing m is spawned from the current location this at a. If the task is spawned within a loop that performs n iterations, the analysis will infer $c(this, a, m) * n$.
3. *Inference of data sizes.* Finally, we need to infer the sizes of the arguments in the task invocations. Typically, size analysis [7] infers upper bounds on the data sizes at the end of the program execution. Here, we are interested in inferring the sizes at the points in which tasks are spawned. In particular, given an instruction a!m(x), we aim at over-approximating the size of x when the program reaches the above instruction. If the above instruction can be executed several times, we aim at inferring the largest size of x, denoted $\alpha(x)$, in all executions of the instructions. Altogether, $c(this, a, m) * \alpha(x)$ is a safe over-approximation of the data size transmission due to such instruction. The analysis will infer such information for each pair of locations in the system that communicate, annotating also the task that was spawned.

We demonstrate the accuracy and feasibility of the presented cost analysis by implementing a prototype analyzer within the SACO system [2], a static analyzer for distributed concurrent programs. Preliminary experiments on some typical applications for distributed programs show the feasibility and accuracy of our analysis. The tool can be used on-line from a web interface available at `http://costa.ls.fi.upm.es/web/saco`.

The remaining of the paper is organized as follows. The next section will present the distribution model that we use to formalize the analysis. Sec. 3 defines the concrete notion of transmission data size that we then want to over-approximate by means of static analysis. Sec. 4 presents the static analysis that carries out the three steps mentioned above. Sec. 5 reports on preliminary experimental results and Sec. 6 concludes.

2 Distribution Model

We consider a distributed programming model with explicit locations and based on the actor-based paradigm [1]. Each location represents a processor with a procedure stack and an unordered queue of pending tasks. Initially all processors are idle. When an idle processor's task queue is not empty, some task is selected for execution. Besides accessing its own processor's global storage, each task can post tasks to the queues of any processor (*message passing*), including its own, and synchronize with the completion of tasks. This synchronization is done by means of *future variables* [8]. When a task completes or when it is awaiting for another task to terminate, its processor becomes idle again, chooses the next pending task, and so on. This distribution model captures the essence of the concurrency model of languages like X10 [12], Erlang [6], Scala [10] or ABS [11].

2.1 Syntax

Regarding data, the language contains basic types B (int, bool ...) and parametric data types D. Data types are declared by listing all the possible constructors C and their arguments, a syntax similar to functional languages like *Haskell*:

(Type variable)	$N ::= a, b, c \dots$
(Basic type)	$B ::= \mathsf{int} \mid \mathsf{bool} \mid \mathsf{void} \mid \dots$
(Data type declaration)	$Dd ::= \mathsf{data}\ D(N_1, \dots, N_n) = C_1 \mid \dots \mid C_k \quad (n \geq 0, k > 0)$
(Constructor)	$C ::= Co(N_1, \dots, N_n) \quad (n \geq 0)$
(Ground type)	$T ::= B \mid D(T_1, \dots, T_n) \quad (n \geq 0)$

Example 1 (Data types). We define integer lists and general binary trees as:
data List = Nil | Cons(int, List)
data Tree(a) = Leaf(a) | Branch(a, Tree(a), Tree(a))
Using the previously declared constructors the list $l = [1, 2, 3]$ is defined as l = Cons(1, Cons(2, Cons(3,Nil))), and the binary tree t with 2 at the root, 1 as left child and 3 as right child as t = Branch(2, Leaf(1), Leaf(3))

Apart from data type declarations, the language allows the definition of functions based on pattern matching as in functional languages—e.g. head, tail, length, etc. This syntax has been omitted for the sake of conciseness, as it does not play an important role for presenting the analysis.

Regarding programs, the number of distributed locations needs not be known a priori (e.g., locations may be virtual). Syntactically, a location will therefore

```
1  main (List l, int s)
        {
2    x = newLoc;
3    y = newLoc;
4    z = newLoc;
5    x!extend(l,s);
6  }
7
8  int foo (int i) {
9    return i;
10 }
11 void extend (List l,int s) {
12   while(s > 0) {
13     Fut f= y!add(l,5);
14     await f?;
15     l = f!get;
16     z!process(l);
17     s = s − 1;
18   }
19 }
20 List add (List l, int e) {
21   List r = Cons(e,l);
22   return r;
23 }
24 void process (List le) {
25   while(le != Nil) {
26     Int h = head(le)
27     y!foo(h);
28     le = tail(le);
29   }
30 }
```

Fig. 1. Running Example

be similar to an *object* and can be dynamically created using the instruction newLoc. The program is composed by a set of methods finished with a return instruction $M ::= T\ m(\bar{T}\ \bar{x})\{s; \text{return } x;\}$ where s takes the form:

$$s ::= s; s \mid x = e \mid x = f.\text{get} \mid \text{if } e \text{ then } s \text{ else } s \mid \text{while } e \text{ do } s \mid b = \text{newLoc}$$
$$\mid f = b!m(\bar{x}) \mid \text{await } f?$$

The notation $\bar{T}$ is used as a shorthand for $T_1, \ldots, T_n$, and similarly for other names. The special location identifier *this* denotes the current location. For the sake of generality, the syntax of expressions e is left open. The semantics of future variables f and concurrency instructions is explained below.

Example 2 (running example). Fig. 1 shows a method main which creates three distributed locations, x, y and z, and receives a list of integers, l, and one integer, s. In the example, we assume that x, y and z are global variables and thus accessible to all methods. Also, we have omitted return instructions in void tasks. Method main spawns task extend at location x in Line 5 (L5 for short) and sends data l and x (thus there is data transmission at this point). Method extend extends l with s new elements. To do this, it invokes method add at location y that extends the list with a new element (L13). The await instruction at L14 awaits for the termination of add. The result is retrieved using the get instruction at L15, where besides we assign the result to l. Within the loop of extend, tasks executing process are spawned at location z. The execution of process traverses the list in the while loop and invokes foo for each element in l. An important point to note is that, besides the data transmitted when asynchronous tasks are spawned, the instruction get also involves data transmission to retrieve the results.

2.2 Semantics

A *program state* has the form $loc_1 \| \ldots \| loc_n$, denoting the currently existing distributed locations. Each *location* is a term $loc(lid, tid, \mathcal{Q})$ where *lid* is the location identifier, *tid* is the identifier of the *active task* which holds the location's lock or $\bot$ if the lock is free, and $\mathcal{Q}$ is the set of tasks at the location. Only one task, which holds the location's *lock*, can be *active* (running) at this location. All

$$\frac{t = tsk(tid, m, l, \langle x = \mathsf{newLoc}; s\rangle),\ \text{fresh}(lid_1)\ ,\ l' = l[x \to lid_1]}{\begin{array}{c} loc(lid, tid, \{t\} \cup \mathcal{Q}) \leadsto \\ loc(lid, tid, \{tsk(tid, m, l', s)\} \cup \mathcal{Q}) \parallel loc(lid_1, \bot, \{\}) \end{array}}\ (\textsc{newLoc})$$

$$\frac{l(x) = lid_1,\ \text{fresh}(tid_1),\ l_1{=}buildLocals(\bar{z}, m_1),\ l' = l[f \to \langle tid_1, \bot, \bot\rangle]}{\begin{array}{c} loc(lid, tid, \{tsk(tid, m, l, \langle f{=}x!m_1(\overline{z}); s\rangle)\} \cup \mathcal{Q}) \parallel loc(lid_1, \_, \mathcal{Q}') \leadsto \\ loc(lid, tid, \{tsk(tid, m, l', s)\} {\cup} \mathcal{Q}) \parallel \\ loc(lid_1, \_, \{tsk(tid_1, m_1, l_1, body(m_1)) \cup \mathcal{Q}'\}) \end{array}}\ (\textsc{async})$$

$$\frac{l(x) = v, l_1(f) = \langle tid, \bot, \bot\rangle, l_1' = l_1[f \to \langle tid, true, \bot\rangle]}{\begin{array}{c} loc(lid, tid, \{tsk(tid, m, l, \langle \mathsf{return}\ x\rangle)\} \cup \mathcal{Q}) \parallel loc(lid_1, \_, \{tsk(tid_1, \_, l_1, \_)\} \cup \mathcal{Q}_1) \leadsto \\ loc(lid, \bot, \{tsk(tid, m, l, \epsilon(v))\} \cup \mathcal{Q}) \parallel loc(lid_1, \_, \{tsk(tid_1, \_, l_1', \_)\} \cup \mathcal{Q}_1) \end{array}}\ (\textsc{return})$$

$$\frac{t = tsk(tid, m, l, \langle \mathsf{await}\ f?; s\rangle), l(f) = \langle tid_1, true, \_\rangle}{loc(lid, tid, \{t\} \cup \mathcal{Q}) \leadsto loc(lid, tid, \{tsk(tid, m, l, s)\} \cup \mathcal{Q})}\ (\textsc{await-t})$$

$$\frac{t = tsk(tid, m, l, \langle \mathsf{await}\ f?; s\rangle), l(f) = \langle tid_1, \bot, \bot\rangle}{loc(lid, tid, \{t\} \cup \mathcal{Q}) \leadsto loc(lid, \bot, \{tsk(tid, m, l, \langle \mathsf{await}\ f?; s\rangle)\} \cup \mathcal{Q})}\ (\textsc{await-f})$$

$$\frac{l(f) = \langle tid_1, true, \bot\rangle, l' = l[x \to v, f \to \langle tid_1, true, v\rangle]}{\begin{array}{c} loc(lid, tid, \{tsk(tid, m, l, \langle x = f.\mathsf{get}; s\rangle)\} \cup \mathcal{Q}) \parallel loc(lid_1, \_, \{tsk(tid_1, \_, l_1, \epsilon(v))\} \cup \mathcal{Q}_1) \leadsto \\ loc(lid, tid, \{tsk(tid, m, l', s)\} \cup \mathcal{Q}) \parallel loc(lid_1, \_, \{tsk(tid_1, \_, l_1, \epsilon(v))\} \cup \mathcal{Q}_1) \end{array}}\ (\textsc{get-r})$$

$$\frac{l(f) = \langle tid_1, true, v\rangle, v \neq \bot, l' = l[x \to v]}{loc(lid, tid, \{tsk(tid, m, l, \langle x = f.\mathsf{get}; s\rangle)\} \cup \mathcal{Q}) \leadsto loc(lid, tid, \{tsk(tid, m, l', s)\} \cup \mathcal{Q})}\ (\textsc{get-l})$$

$$\frac{select(\mathcal{Q}) = tid,\ t = tsk(tid, \_, \_, s){\in}\mathcal{Q}, s \neq \epsilon(v)}{loc(lid, \bot, \mathcal{Q}){\leadsto}loc(lid, tid, \mathcal{Q})}\ (\textsc{select})$$

Fig. 2. (Summarized) Semantics for Distributed Execution

other tasks are *pending*, waiting to be executed, or *finished*, if they terminated and released the lock. A *task* is a term $tsk(tid, m, l, s)$ where *tid* is a unique task identifier, m is the name of the method executing in the task, l is a mapping from local variables to their values and s is the sequence of instructions to be executed or $s = \epsilon(v)$ if the task has terminated with value v.

The execution of a program starts from a method m in an initial state S_0 with a single (initial) location with identifier 0 executing task 0 of the form $S_0{=}loc(0, 0, \{tsk(0, \mathsf{m}, l, body(\mathsf{m}))\})$. Here, l maps parameters to their initial values and local references to null (standard initialization), and $body(\mathsf{m})$ refers to the sequence of instructions in the method m. The execution proceeds from the initial state S_0 by selecting *non-deterministically* one of the locations and applying the semantic rules depicted in Fig. 2. The treatment of sequential instructions is standard and thus omitted. The operational semantics $\leadsto$ is given in a rewriting-based style where at each step a subset of the state is rewritten according to the rules as follows. In newloc, the active task *tid* at location *lid* creates a location lid_1 which is introduced to the state with a free lock. async spawns a new task (the initial state is created by *buildLocals*) with a fresh task identifier tid_1 which is added to the queue of location lid_1—the case $lid{=}lid_1$ is analogous, the new task tid_1 is simply added to the queue $\mathcal{Q}$ of lid. The future variable f allows synchronizing the execution of the current task with the completion of the created task, and retrieving its result. The association of the future variable to the task is stored in the local variables table $l'(f){=}\langle tid_1, \bot, \bot\rangle$: the future variable f is linked to task tid_1, the task has not terminated yet (first $\bot$ in the tuple), and the result of the invocation is not available yet (second $\bot$). The rule RETURN is used when a task tid executes a return instruction. The terminating task tid finishes the execution with value v (its sequence of instructions is set to $\epsilon(v)$) and the calling task tid_1 is notified that tid has terminated by setting to $true$ the termination flag of the corresponding future variable—the case $lid{=}lid_1$ is analogous, but storing the termination flag in a task in queue $\mathcal{Q}$. In AWAIT-T, the future variable we are awaiting for points to a finished task (it has the termination flag set to $true$ in the future variable f stored in the local variable table l) and await can be completed. Otherwise, AWAIT-F yields the lock so that any other task of the same location can take it. The rule GET-R retrieves the returning value from the task tid_1 linked to the future variable f, if the corresponding task has terminated and the value has not been retrieved before. If tid_1 has not terminated, it will wait for the value without yielding the lock. If the returning value has been retrieved from the remote object already, it is copied locally from the future variable f by means of GET-L. Finally, in rule SELECT an idle location takes a non-finished task to continue the execution—the function $select(\mathcal{Q})$ non-deterministically returns a task identifier occurring in $\mathcal{Q}$.

Example 3 (semantics). The following sequence is the beginning of a trace of the program in Fig. 1 starting from main(Cons(1,Cons(2,Nil)),7). For the sake of conciseness we represent lists with square brackets—[1,2]—instead of constructors and we use l_e, l_a and l_p to denote initial local mappings, stressing only the important changes to them at each step.

$$\begin{aligned}
S_0 &\equiv loc(0,0,\{tsk(0,\mathsf{main},l_m,\langle \mathsf{x = newLoc};\ldots\rangle)\}) \rightsquigarrow^{\text{NEWLOC}\times 3}\\
S_3 &\equiv loc(0,0,\{tsk(0,\mathsf{main},l_m[\mathsf{x}\mapsto 1,\mathsf{y}\mapsto 2,\mathsf{z}\mapsto 3],\langle \mathsf{x!extend(l,s)}\rangle)\}) \parallel loc(1,\bot,\{\})\\
&\quad \parallel loc(2,\bot,\{\}) \parallel loc(3,\bot,\{\}) \rightsquigarrow^{\text{ASYNC}}\\
S_4 &\equiv loc(0,0,\ldots) \parallel loc(1,\bot,\{tsk(1,\mathsf{extend},l_e,\langle \mathsf{while}\ (s>0)\{\ldots\}\rangle)\})\\
&\quad \parallel loc(2,\bot,\{\}) \parallel loc(3,\bot,\{\}) \rightsquigarrow^{\text{SELECT}} S_5 \rightsquigarrow\\
S_6 &\equiv loc(0,0,\ldots) \parallel loc(1,1,\{tsk(1,\mathsf{extend},l_e,\langle \mathsf{Fut\ f{=}y!add(l,5)};\ldots\rangle)\})\\
&\quad \parallel loc(2,\bot,\{\}) \parallel loc(3,\bot,\{\}) \rightsquigarrow^{\text{ASYNC}}\\
S_7 &\equiv loc(0,0,\ldots) \parallel loc(1,1,\{tsk(1,\mathsf{extend},l_e[\mathsf{f}\mapsto\langle 2,\bot,\bot\rangle],\langle \mathsf{await\ f?};\ldots\rangle)\})\\
&\quad \parallel loc(2,\bot,\{tsk(2,\mathsf{add},l_a,\langle \mathsf{List\ r = Cons(e,l);return\ r}\rangle)\}) \parallel loc(3,\bot,\{\}) \rightsquigarrow^{\text{SELECT}}
\end{aligned}$$

$$\begin{aligned}
S_8 &\equiv loc(0,0,\ldots)) \parallel loc(1,1,\{tsk(1,\mathsf{extend},l_e,\langle \mathsf{await\ f?};\ldots\rangle)\})\\
&\quad \parallel loc(2,2,\{tsk(2,\mathsf{add},l_a,\langle \mathsf{List\ r = Cons(e,l);return\ r}\rangle)\}) \parallel loc(3,\bot,\{\}) \rightsquigarrow S_9 \rightsquigarrow^{\text{RETURN}}\\
S_{10} &\equiv loc(0,0,\ldots) \parallel loc(1,1,\{tsk(1,\mathsf{extend},l_e[\mathsf{f}\mapsto\langle 2,true,\bot\rangle],\langle \mathsf{await\ f?};\ldots\rangle)\})\\
&\quad \parallel loc(2,\bot,\{tsk(2,\mathsf{add},l_a,\epsilon([5,1,2]))\}) \parallel loc(3,\bot,\{\}) \rightsquigarrow^{\text{AWAIT-T+GET-R}}\\
S_{12} &\equiv loc(0,0,\ldots) \parallel loc(2,\bot,\{tsk(2,\mathsf{add},l_a,\epsilon([5,1,2]))\}) \parallel loc(3,\bot,\{\}) \parallel loc(1,1,\\
&\quad \{tsk(1,\mathsf{extend},l_e[\mathsf{f}{\mapsto}\langle 2,true,[5,1,2]\rangle,\mathsf{l}{\mapsto}[5,1,2]],\langle \mathsf{z!process(l)};\ldots\rangle)\}) \rightsquigarrow^{\text{ASYNC}}\\
S_{13} &\equiv loc(0,0,\ldots) \parallel loc(2,\bot,\ldots) \parallel loc(3,\bot,\{tsk(3,\bot,l_p,body(\mathsf{process}))\})\\
&\quad loc(1,1,\{tsk(1,\mathsf{extend},l_e,\langle \mathsf{s = s - 1};\ldots\rangle)\})
\end{aligned}$$

From state S_0 to S_3 we create the three locations x(1), y(2) and z(3) applying rule NEWLOC. In S_3 a new task extend is spawned using rule ASYNC, that is placed in the queue of location 1. Since location 1 is idle but the queue contains the non-finished task 2 in S_4, it takes the lock (SELECT) and executes the first iteration of the loop. In S_6 and S_7 a new task add is spawned to location 2 and it takes the lock. Note that in S_7 the local mapping is extended to store that the future variable f is linked to task 2, which is not finished yet ($\perp$). Task 2 finishes immediately by assigning variable r and returning: it stores the final value [5,1,2] and notifies task 1 (RETURN). Since task 2 is finished in S_{10} the await and get instructions can proceed (rules AWAIT-T and GET-R resp.), yielding to S_{12}. Finally, task 2 spawns a new task process in location 3.

3 The Notion of Transmission Data Size

The *transmission data size* of a program execution is the total amount of data that is moved between locations. There are two situations that generate data movement between locations: a) when a task is invoked (in this case it sends a message to the destination location containing all the arguments); and b) when the returning value of a task invocation is retrieved (it sends a message containing that value). Therefore, only these two transitions of states will contribute to the transmission data size of a program execution. In order to define this notion we will consider that state transitions are decorated with transmission data size information: $S_1 \leadsto^{d}_{(lid_1,lid_2,m)} S_2$, meaning a transmission of d units of data from object lid_1 to lid_2 through m. Transitions that do not generate data transmission will be decorated as $S_1 \leadsto^{0}_{\epsilon} S_2$. Since we are considering an abstract representation of data by means of functional types, we will focus on *units of data* transmitted instead of bits, which depends on the actual implementation and is highly platform-dependent. Concretely, we assume that the cost of transmitting a basic value or a data type constructor is one unit of data. This size measure is known as *term size.* However, the static analysis we propose later would work also with any other mapping from data types to corresponding sizes (given by means of a function α such as the one below).

Definition 1 (term size). *The* term size *of value v—$\alpha(v)$—is defined as:*

$$\alpha(v) = \begin{cases} 1 + \sum_{i=1}^{n} \alpha(v_i) & \text{if } v = Co(v_1 \dots v_n), \\ 1 & \text{otherwise.} \end{cases}$$

Example 4 (size measures). Considering the term size measure, the size of the list l = Cons(1, Cons(2, Cons(3,Nil))) is $\alpha(\mathsf{l}) = 7$ (4 data constructors and 3 integers) and the size of the tree t = Branch(2, Leaf(1), Leaf(3)) is $\alpha(\mathsf{t}) = 6$ (3 constructors plus 3 integers).

Definition 2 (decorated step). *A step $S_1 \leadsto S_2$ using rule R from Fig. 2 is decorated as follows:*

– *If $R =$* ASYNC *then the step is decorated as $S_1 \rightsquigarrow^{d}_{(lid,lid_1,m)} S_2$, where $d = \mathcal{I} + \sum_{z \in \overline{z}} \alpha(l(z))$, and m is the method invoked in the call. The constant $\mathcal{I}$ is the size of establishing the communication, and we add the size of all the arguments passed to the destination location. Note that a task invocation inside the same location ($lid = lid_1$) will not generate any transmission, so in these cases the decoration is $S_1 \rightsquigarrow^{0}_{\epsilon} S_2$.*
– *If $R =$* GET-R *then the decorated step is $S_1 \rightsquigarrow^{d}_{(lid_2,lid_1,m)} S_2$, where $d = \mathcal{I} + \alpha(v)$, v corresponds to the returned value, and m is the method that returned v. As before, if $lid = lid_1$ then there is no transmission and the decoration is $S_1 \rightsquigarrow^{0}_{\epsilon} S_2$.*
– *If $R \in \{$*NEWLOC, RETURN, AWAIT-T, AWAIT-F, GET-L, SELECT*$\}$, then the step does not move any data, so it is decorated with an empty label: $S_1 \rightsquigarrow^{0}_{\epsilon} S_2$.*

Observe that rules AWAIT-T, AWAIT-F and GET-L use local variables only, and therefore do not perform any remote communication. Rule RETURN notifies the termination of a method to the caller location, although its cost is included in the size $\mathcal{I}$ for establishing the communication included in rule ASYNC.

Definition 3 (transmission data size of a trace). *Given a decorated trace $\mathcal{T} \equiv S_0 \rightsquigarrow^{d_1}_{o_1} S_1 \rightsquigarrow^{d_2}_{o_2} \dots \rightsquigarrow^{d_n}_{o_n} S_n$, the transmission data size of $\mathcal{T}$—$trans(\mathcal{T})$—is defined as:*

$$trans(\mathcal{T}) = \sum_{i=1}^{n} d_i$$

Example 5 (transmission data size). The decorated trace from Ex. 3 is:

$$\mathcal{T}_d \equiv S_0 \rightsquigarrow^{0}_{\epsilon} S_1 \rightsquigarrow^{0}_{\epsilon} S_2 \rightsquigarrow^{0}_{\epsilon} S_3 \rightsquigarrow^{\mathcal{I}+6}_{(0,1,\mathsf{extend})} S_4 \rightsquigarrow^{0}_{\epsilon} S_5 \rightsquigarrow^{0}_{\epsilon} S_6 \rightsquigarrow^{\mathcal{I}+6}_{(1,2,\mathsf{add})} S_7 \rightsquigarrow^{0}_{\epsilon} S_8 \rightsquigarrow^{0}_{\epsilon} S_9 \rightsquigarrow^{0}_{\epsilon} S_{10} \rightsquigarrow^{0}_{\epsilon} S_{11} \rightsquigarrow^{\mathcal{I}+7}_{(2,1,\mathsf{add})} S_{12} \rightsquigarrow^{\mathcal{I}+7}_{(1,3,\mathsf{process})} S_{13}$$

From S_3 to S_4 it sends a message ($\mathcal{I}$) from location 0 to 1 containing the arguments of the call: l=Cons(1,Cons(2,Nil)) and s=7, where $\alpha(l) = 5$ and $\alpha(7) = 1$. Similarly, from S_6 to S_7 it sends a message from location 1 to 2 with the arguments l and 5 for task add. In State S_9 it executes a return instruction, that notifies the termination to the caller, but its size is already considered in the call (S_6). The returning value from the call to add is actually received from the caller at S_{12}, by means of a message from location 2 to 1 with the returning value r = Cons(5,Cons(1,Cons(2,Nil))), $\alpha(r) = 7$. Finally, the invocation of task process in state S_{12} sends a message from location 1 to 3 containing the argument l = Cons(5,Cons(1,Cons(2,Nil))), of size 7. Considering this decorated trace, the total transmission data size is:

$$trans(\mathcal{T}_d) = (\mathcal{I} + 6) + (\mathcal{I} + 6) + (\mathcal{I} + 7) + (\mathcal{I} + 7) = 4{*}\mathcal{I} + 26$$

In other words, the transmission data size is $4{*}\mathcal{I}$ units of data for creating 4 messages, and 26 units of data for the transmission of values.

The transmission data size of a trace takes into account all the invocation and returning messages, independently of the location involved. In our setting we have several locations that can be executing in different machines or CPUs, so it is interesting to limit transmission data size to some locations. We define a *restriction* operator over traces to consider only data-moving steps between certain locations.

Definition 4 (trace restriction). *Given a decorated trace $\mathcal{T}$, two location identifiers, l_1 and l_2, a method m, the trace restriction $\mathcal{T}|_{l_1 \xrightarrow{m} l_2}$ is defined as:*

$$\mathcal{T}|_{l_1 \xrightarrow{m} l_2} = \{S_{i-1} \rightsquigarrow^{d_i}_{(l_1,l_2,m)} S_i \mid S_{i-1} \rightsquigarrow^{d_i}_{(l_1,l_2,m)} S_i \in \mathcal{T}\}$$

4 Automatic Inference of Transmission Data Sizes

The analysis has three main parts which are introduced in the following sections: Sec. 4.1 is encharged of inferring the locations in the distributed system and using them to define the *cost centers* on which the cost analysis is based; Sec. 4.2 infers upper bounds on the number of tasks spawned along any execution of the program; Sec. 4.3 over-approximates the sizes of the data transmitted when spawning asynchronous calls and when retrieving their results.

4.1 Inference of Distributed Locations

Since locations can be dynamically created, we need an analysis that abstracts them into a *finite* abstract representation, and that tells us which (abstract) location a reference variable is pointing-to. *Points-to* analysis [14,13,15] solves this problem. It infers the set of memory locations that a reference variable can *point-to*. Different abstractions can be used and our method is parametric on the chosen abstraction. Any points-to analysis that provides the following information with more or less accurate precision can be used (our implementation uses [13]): (1) $\mathcal{O}$, the set of abstract locations; (2) a function $pt(pp, v)$ that, for a given program point pp and variable v, returns the set of abstract locations in $\mathcal{O}$ to which v may point.

Example 6 (distributed locations). Consider the main method shown in Fig. 1 which creates three locations x, y and z at L2, L3 and L4, and which are abstracted, respectively, as o_x, o_y and o_z. By using the points-to analysis we obtain the following set of objects created along the execution of main, $\mathcal{O} = \{o_x, o_y, o_z\}$. Besides, the points-to analysis can infer information for the local variables at the level of program point, that is, $pt(L11, \mathsf{this}) = \{o_x\}$, $pt(L13, \mathsf{y}) = \{o_y\}$, $pt(L16, \mathsf{z}) = \{o_z\}$, $pt(L20, \mathsf{this}) = \{o_y\}$, $pt(L24, \mathsf{this}) = \{o_z\}$, $pt(L26, \mathsf{y}) = \{o_y\}$ or $pt(L8, \mathsf{this}) = \{o_y\}$.

The distributed locations that the points-to analysis infers are used to define the *cost center*s [3] that the resource analysis will use. The notion of cost center is used to attribute the cost of each instruction to the location that executes it. In the above example, we have three locations which lead to three cost centers, $c(o_x)$, $c(o_y)$ and $c(o_z)$.

4.2 Inference of Number of Tasks Spawned

Our analysis builds upon well-established work on cost analysis [9,16,3]. Such analyses are based on a generic notion of resource which can be instantiated to measure different metrics such as number of executed instructions, amount of memory created, number of calls to methods, etc. In particular, the *cost model* is used to determine the type of resource we are measuring. Traditionally, a cost model is a function $\mathcal{M} : Instr \rightarrow \mathbb{N}$ which, for each instruction in the program, returns a natural number which represents its cost. As examples of cost models we could have: for counting the number of instructions executed by a program, the cost model counts one unit for any instruction, i.e., $\mathcal{M}^i(ins) = 1$; for counting the number of calls, we can use $\mathcal{M}^c(ins) = 1$ if $ins \equiv x!m(\_)$; and 0 otherwise. When the analysis uses cost centers, the cost model additionally defines to which cost center the cost must be attributed. For instance, when counting number of instructions, we have that $\mathcal{M}(i) = \sum_{o \in pt(pp,this)} c(o)*1$, where pp is the program point of instruction i, i.e., the instruction is accumulated in all locations that it can be executed (this is given by the locations to which the this reference can point).

In what follows, we use the cost analyzer as a black box in the following way. Given a method $m(\bar{x})$ and a cost model, the cost analyzer gives us an *upper bound* for the total cost (for the resource specified in the cost model) of executing m of the form $\mathcal{U}_m(\bar{x}) = \sum_{i=1}^{n} cc_i{*}C_i$, where cc_i is a cost center and C_i is a cost expression that bounds the cost of the computation carried out by the cost center cc_i. If one is interested in studying the computation performed by one particular cost center cc_j, we simply replace all cc_i with $i \neq j$ by 0 and cc_j by 1. In order to obtain the cost expression C_i, the cost analyzer needs to over-approximate the number of iterations that loops perform, and infer the maximum sizes of data. For the sake of this paper, we do not need to go into the technical details of this process. To infer an upper bound on the number of tasks spawned by the program, we simply have to define a *number of tasks cost model* and use the cost analyzer as a black box.

Definition 5 (number of tasks cost model). *Given an instruction ins at program point pp, we define the number of tasks cost model, $\mathcal{M}^t(ins)$ as a function which returns $c(o_1, o_2, m)$ if $ins \equiv f{=}y!m(\_) \wedge o_1 \in pt(pp, this) \wedge o_2 \in pt(pp, y) \wedge o_1 \neq o_2$, and 0 otherwise.*

The main feature of the above cost model is that we use an extended form of cost centers which are triples of the form $c(o_1, o_2, m)$, where o_1 is the object that is executing, o_2 is the object responsible for executing the call, and m is the name of the invoked method. These cost centers are symbolic expressions that will be part of the upper bound computed by the analyzer. Let us see an example.

Example 7 (number of tasks). For the code in Fig. 1, cost analysis infers that the number of iterations of the loop in extend (at L12) is bounded by the expression $\mathsf{nat}(s)$, where $\mathsf{nat}(e)$ returns e if $e > 0$ and 0 otherwise. Since the size of l is increased within the loop at L12, the maximum number of iterations for the loop at L25 is produced in the last call to process. Recall that l represents the term size of the list l (see Def. 1), and it counts 2 units for each element in the list. Therefore, each iteration of the loop at L25 increments the term size of the list in 2 units and, consequently, the last call to process is done with a list of size $l+2*s$. The loop in process (L25) traverses the list received as argument consuming 2 size units per iteration. Therefore, the expression $(l + 2 * s)/2 = l/2 + s$ bounds the number of iterations of such loop. As process is called $\mathsf{nat}(s)$ times, $\mathsf{nat}(s) * \mathsf{nat}(l/2 + s)$ bounds the number of times that the body of the loop at L25 is executed. Then, by applying the number of tasks cost model we obtain the following expression that bounds the number of tasks spawned:

$$\begin{aligned}\mathcal{U}^t_{\mathsf{extend}}(l, s) = {} & c(o_x, o_y, \mathsf{add}) * \mathsf{nat}(s) + \\ & c(o_x, o_z, \mathsf{process}) * \mathsf{nat}(s) + \\ & c(o_z, o_y, \mathsf{foo}) * (\mathsf{nat}(s) * \mathsf{nat}(l/2{+}s))\end{aligned}$$

From the upper bounds on the tasks spawned, we can obtain a range of useful information: (1) If we are interested in the number of communications for the whole program, we just replace all expressions $c(o_1, o_2, m)$ by 1. (2) Replacing all cost centers of the form $c(o, \_, \_)/c(\_, o, \_)$ by 1 for the object o and the remaining ones by 0, we obtain an upper-bound on the number of tasks spawned from/in o. We use, respectively, $\mathcal{U}_m|_{o\rightarrow}$ and $\mathcal{U}_m|_{\rightarrow o}$ to refer to the UB on the outgoing/incoming tasks. (3) Replacing $c(o_1, o_2, \_)$ by 1 for selected objects and the remaining ones by 0, we can see the tasks spawned by o_1 in o_2, denoted by $\mathcal{U}_m|_{o1\rightarrow o_2}$. (4) If we are interested in a particular method p, we can replace $c(\_, \_, p)$ by 1 and the rest by 0, we use $\mathcal{U}_m|_{\xrightarrow{p}}$ to denote it.

Example 8 (number of tasks restriction). Given the upper bound of Ex. 7, the number of tasks spawned from o_x to o_y is captured by replacing $c(o_x, o_y, \_)$ (the method is not relevant) by 1 and the rest by 0. Then, we obtain the expression $\mathcal{U}^t_{\text{extend}}|_{o_x\rightarrow o_y} = \mathsf{nat}(s)$, which shows that we have one task for each iteration of the loop at L13. We can also obtain an upper bound on the number of tasks from o_z to o_y, $\mathcal{U}^t_{\text{extend}}|_{o_z\rightarrow o_y} = \mathsf{nat}(s)*\mathsf{nat}(l/2{+}s)$. The number of tasks spawned using method foo are captured by $\mathcal{U}^t_{\text{extend}}|_{\xrightarrow{\mathsf{process}}} = \mathsf{nat}(s)$.

4.3 Inference of Amount of Transmitted Data

Our goal now is to infer, not only the number of tasks spawned, but also the sizes of the arguments in the task invocation and of the returned values. Formally, this is done by extending the previous cost model to include data sizes as well. We rely on two auxiliary functions. Given a variable x at a certain program point, function $\alpha(x)$ returns the term size of this variable at this point, as defined in Sec. 3. Besides, after spawning a task, we are interested in knowing whether the result of executing the task is retrieved, and in such case we accumulate the size of the return value. This information is computed by a *may-happen-in-parallel* analysis [5] which allows us to know to which task a future variable is associated. Thus, we can assume the existence of a function *hasGet*(*pp*) which returns if the result of the task spawned at program point *pp* is retrieved by a get instruction.

Now, we define a new cost model that counts the sizes of the data transferred in each communication by relying on the two functions above.

Definition 6 (data sizes cost model). *Given a program point pp we define the cost model $\mathcal{M}^d(ins)$ as a function which returns $sc(ins)$ if $pp : ins \equiv r = y!m(\overline{x}) \wedge o_1 \neq o_2 \wedge o_1 \in pt(pp, this) \wedge o_2 \in pt(pp, y)$, and 0, otherwise; where*

$$sc(ins)=\begin{cases} c(o_1,o_2,m)*(\mathcal{I}+\sum_{x_i\in\overline{x}} \mathsf{nat}(\alpha(x_i)))+c(o_2,o_1,m)*(\mathcal{I}+\mathsf{nat}(\alpha(r))) & \textit{if } hasGet(pp) \\ c(o_1,o_2,m)*(\mathcal{I}+\sum_{x_i\in\overline{x}} \mathsf{nat}(\alpha(x_i))) & \textit{otherwise} \end{cases}$$

Observe that the above cost model extends the one in Def. 5 as it extends the number of tasks cost model with the sizes of the data transmitted. Intuitively, as any call always transfers its input arguments, their size is always included (second case). However, the size of the returned information is only included when there exists a get instruction that retrieves this information (first case). In each case, we include the size for sending the messages $\mathcal{I}$. Note that the cost centers reflect the direction of the transmission, $c(o_1, o_2, m)$ corresponds to a transmission from o_1 to o_2 through a call to m, whereas $c(o_2, o_1, m)$ corresponds to the information returned by o_2 in response to a call to m spawned by o_1. If needed, call and return cost centers can be distinguished by marking the method name, e.g., m for calls and m^r for returns. As already mentioned, nat denotes the positive value of an expression. We wrap the size of each argument using nat because this way the analyzer treats them as an expression whose cost we want to maximize (the technical details of the maximization operation can be found in [4]). Therefore, the upper bound inferred by the analyzer using this cost model already provides the overall information (i.e., number of tasks spawned and maximum size of the data transmitted).

Example 9 (data sizes cost model). Let us see the application of the cost model to the calls at L16, L13 and L26. At L16 we have the instruction z!process(l). As the program does not retrieve any information from process(l), the function $hasGet(L16)$ returns false, and thus we only include the calling data. Then, using the points-to information in Ex. 6, the application of $\mathcal{M}^d$ at L16 returns: $\mathcal{M}^d(\mathsf{z!process(l)}) = c(o_x, o_z, \mathsf{process}) * \mathcal{I} + \mathsf{nat}(\alpha(l))$. As l is a data structure and it is modified within the loop, $\alpha(l)$, returns the term size of l. Observe that the expression captures, not only the objects and the method involved in the call within the cost center, but also the amount of data transferred in the call, $\mathsf{nat}(\alpha(l))$. The application of $\mathcal{M}^d$ to the call at L13, f = y!add(l,5), returns the expression:

$$\begin{aligned}\mathcal{M}^d(\mathsf{f{=}y!add(l_0{,}5)}) = {} & c(o_x, o_y, \mathsf{add}) * (\mathcal{I} + \mathsf{nat}(\alpha(l_0)) + \mathsf{nat}(\alpha(5)))+ \\ & c(o_y, o_x, \mathsf{add}) * (\mathcal{I} + \mathsf{nat}(\alpha(f)))\end{aligned}$$

In this case, at L15 we have a get for the call at L13, so $hasGet(L13) = true$. Note that we use l_0 to refer to the value of l at the beginning of the loop and l to refer to the value of the list after calling add. The application of $\alpha(5)$ returns

1, as it is a basic type (counting as one constructor). The call at L27 returns the expression $c(o_y, o_z, \mathsf{foo}) * (\mathcal{I} + \mathsf{nat}(\alpha(h)))$.

As we have explained above, the size of a data structure might depend on the input arguments that in turn can be modified along the program execution. Consequently, if we are in a loop, for the same program point, the amount of data transferred in one call can be different for each iteration of the loop. Soundness of the cost analysis ensures that it provides the worst possible size in such case. Technically, it is done by maximizing [4] the expressions inside nat within their calling context.

Example 10 (data sizes upper bound). Once the cost model is applied to all instructions in the program, we obtain a set of recursive equations which define the transmission data sizes within the locations in the program. After solving such equations using [4], we obtain the following expression which defines the transmission data sizes of any execution starting from extend, denoted by $\mathcal{U}^d_{\mathsf{extend}}$:

$$\begin{aligned}
\mathcal{U}^d_{\mathsf{extend}}(l, s) = {} & c(o_x, o_y, \mathsf{add}) * \mathsf{nat}(s) * (\mathcal{I} + \mathsf{nat}(l + s * 2 - 2) + 1) + & \text{①} \\
& c(o_y, o_x, \mathsf{add}) * \mathsf{nat}(s) * (\mathcal{I} + \mathsf{nat}(l + s * 2)) + & \text{②} \\
& c(o_x, o_z, \mathsf{process}) * \mathsf{nat}(s) * (\mathcal{I} + \mathsf{nat}(l + s * 2)) + & \text{③} \\
& c(o_z, o_y, \mathsf{foo}) * (\mathsf{nat}(s) * \mathsf{nat}(l/2{+}s)) * (\mathcal{I} + 1) & \text{④}
\end{aligned}$$

The expression at ① includes the transmission from o_x to o_y. The worst case size of the list at this point is $\mathsf{nat}(l+s*2-2)$, this is because initially the list has size $\mathsf{nat}(l)$ and at each iteration of the loop, the size is increased in method add by two elements: Cons and an integer value. As the loop performs s iterations, in the last invocation to add it has length $l + (s - 1) * 2$. This size is assumed for all loop iterations (worst case size), hence we infer that the maximum data size transmitted from o_x to o_y is $\mathsf{nat}(s) * (\mathcal{I} + \mathsf{nat}(l + s * 2 - 2) + 1)$, the 1 is due to the second argument of the call (an integer). At ②, o_x receives from o_y the same list, but including the last element, that is $\mathsf{nat}(l + s * 2)$. The same list is obtained at ③. In ④, the cost is constant in all iterations (1 integer).

As already mentioned in Sec. 4.2, the fact that cost centers are symbolic expressions allows us to extract different pieces of information regarding the amount of data transferred between the different abstract locations involved in the communications. With $\mathcal{U}^d$ we can infer, not only an upper-bound on the total amount of data transferred along the program execution, but also the size of the data transferred between two objects, or the incoming/outgoing data sent/received by a particular object.

Example 11 (data sizes restriction). From $\mathcal{U}^d_{\mathsf{extend}}(l, s)$, using the cost centers as we have explained in Ex. 8, we can extract different types of information about the data transferred. For instance, we can bound the size of the outgoing data from location x:

$$\mathcal{U}^d_{\mathsf{extend}}(l, s)|_{o_x \rightarrow} = \mathsf{nat}(s) * (\mathcal{I} + \mathsf{nat}(l + s * 2 - 2) + 1) + \mathsf{nat}(s) * (\mathcal{I} + \mathsf{nat}(l + s * 2))$$

Or the incoming data sizes for the location y:

$$\mathcal{U}^d_{\mathsf{extend}}(l, s)|_{\rightarrow o_y} = \mathsf{nat}(s) * (\mathcal{I} + \mathsf{nat}(l + s * 2 - 2) + 1) + (\mathsf{nat}(s) * \mathsf{nat}(l/2{+}s)) * (\mathcal{I} + 1)$$

Table 1. Experimental results (times in ms)

				Nodes			Methods			Pairs		
Benchmark	loc	$\#_c$	T	$\%^n_M$	$\%^n_m$	$\%^n_a$	$\%^m_M$	$\%^m_m$	$\%^m_a$	$\%^p_M$	$\%^p_m$	$\%^p_a$
BBuffer	200	17	829	25.7	0.6	16.3	43.9	0.1	6.2	7.3	0.0	0.7
MailServer	119	13	693	30.0	4.4	15.4	27.3	0.5	10.0	8.7	0.0	0.6
Chat	302	10	171	40.5	7.5	20.0	12.7	0.1	3.0	9.6	0.0	1.1
DistHT	146	9	1204	48.0	3.0	18.7	40.7	0.3	10.0	8.0	0.0	0.9
BookShop	366	10	3327	58.7	3.9	23.9	23.6	0.1	8.3	29.5	0.0	1.5
PeerToPeer	263	19	62575	27.7	0.1	15.6	20.6	0.1	5.8	5.9	0.0	0.5

Theorem 1 (soundness). *Let P be a program and l_1, l_2 location identifiers. Let $\mathcal{O}$ be the object names computed by a points-to analysis of P. Let o_1, o_2 be the abstractions of l_1, l_2 in $\mathcal{O}$. Then, given a trace $\mathcal{T}$ from P with arguments $\overline{x}$ we have that*

$$trans(\mathcal{T}|_{l_1 \xrightarrow{m} l_2}) \leq \mathcal{U}^d_P(\overline{x})|_{o_1 \xrightarrow{m} o_2}.$$

5 Experimental Results

We have implemented our analysis in SACO [2] and applied it to some typical examples of distributed systems: BBuffer, a bounded-buffer for communicating several producers and consumers; MailServer, a client-server distributed system; Chat, a chat application; DistHT, a distributed hash table; BookShop, a web shop client-server application; and PeerToPeer, a peer-to-peer network with a set of interconnected peers. Experiments have been performed on an Intel Core i7 at 3.4GHz with 8GB of RAM, running Ubuntu 12.04.

We have applied our analysis and evaluated the upper bound expressions for different combinations of concrete input values so as to obtain some quantitative information about the analysis. Table 1 summarizes the results obtained. Columns Benchmark and loc show, resp. the name and the number of program lines of the benchmark. Column $\#_c$ displays the number of locations identified by the analysis. Column T shows the time to perform the inference of the transmission data sizes. We have studied the transmission data sizes among each pair of locations identified by the points-to analysis. We have studied data transmission from three points of view: (1) from a location with the rest of the program, (2) from a method, and (3) among pairs of locations. In case (1), we try to identify potential bottlenecks in the communication, i.e., those locations that produce/consume most of the data in the benchmark. Also, we want to observe locations that do not have much communication. In the former, such locations should have a fast communication channel, while in the latter we can still have a good response time with slower bandwidth conditions. Columns $\%^n_M$, $\%^n_m$, $\%^n_a$ show, respectively, the percentage of the location that accumulates more traffic (incoming + outgoing) w.r.t. the total traffic in the system, for the location with less traffic, and the average for the traffic of all locations. Similarly, columns $\%^p_M$, $\%^p_m$, $\%^p_a$ show, for case (3), which is the percentage of the total traffic transmitted by the pair of locations that have more traffic, by the pair with less traffic and the average between the traffic of all pairs, respectively. Finally, regarding case (2), columns under Methods show similar information but taking into account the task that performs the communication, i.e., the percentage of the traffic transmitted by the task that transmits more (resp., less) amount of data, $\%^m_M$ (resp., $\%^m_m$), and the average of the transmissions performed by each task ($\%^m_a$).

We can observe in the table that our analysis is performed in a reasonable time. One important issue is that we only have to perform the analysis once, and the information can be extracted later by evaluating the upper bound with different parameters and focusing in the communications of interest. In the columns for the locations, we can see that all benchmarks are relatively well distributed. The average of the data transmitted per location is under 25% for all benchmarks. BookShop is the benchmark which could have a communication bottleneck as it accumulates in a single location 58.7% of the total traffic. Regarding methods, it is interesting to see that for all benchmarks no method accumulates more than 45% of the total traffic. Moreover, the table shows that in all benchmarks there is at least one method that requires less than 0.5%, in most cases this method (or methods) is an object constructor. Regarding pairs of locations, in all benchmarks there is at least one pair of locations that do not communicate, $\%^p_m = 0$ for all benchmarks. This is an expected result, as it is quite often to have pairs of locations which do not communicate in a distributed program. Our experiments thus confirm that transmission among pairs of locations is relatively well distributed, as in most benchmarks, except for BookShop, the pair with highest traffic requires less than 10% of the total traffic.

6 Conclusions

We have presented a static analysis to soundly approximate the amount of data transmitted among the locations of a distributed system. This is an important contribution to be able to infer the response times of distributed components. In particular, if one knows the bandwidth conditions among each pair of locations, we can infer the time required to transmit the data and to retrieve the result. This time should be added to the time required to carry out the computation at each location, which is an orthogonal issue. Conversely, we can use our analysis to establish the bandwidth conditions required to ensure a certain response time. Technically, our analysis is formalized by defining a new cost model which captures only the data transmission aspect of the application. This cost model can be plugged into a generic cost analyzer for distributed systems, that directly returns an upper bound on the transmission data sizes, without requiring any modification to the other components of the cost analyzer.

Acknowledgments. This work was funded partially by the EU project FP7-ICT-610582 ENVISAGE: Engineering Virtualized Services (http://www.envisage-project.eu) and by the Spanish projects TIN2008-05624 and TIN2012-38137.

References

1. Agha, G.: Actors: A Model of Concurrent Computation in Distributed Systems. MIT Press, Cambridge (1986)
2. Albert, E., Arenas, P., Flores-Montoya, A., Genaim, S., Gómez-Zamalloa, M., Martin-Martin, E., Puebla, G., Román-Díez, G.: SACO: Static Analyzer for Concurrent Objects. In: Ábrahám, E., Havelund, K. (eds.) TACAS 2014. LNCS, vol. 8413, pp. 562–567. Springer, Heidelberg (2014)
3. Albert, E., Arenas, P., Genaim, S., Gómez-Zamalloa, M., Puebla, G.: Cost Analysis of Concurrent OO programs. In: Yang, H. (ed.) APLAS 2011. LNCS, vol. 7078, pp. 238–254. Springer, Heidelberg (2011)
4. Albert, E., Arenas, P., Genaim, S., Puebla, G.: Closed-Form Upper Bounds in Static Cost Analysis. Journal of Automated Reasoning 46(2), 161–203 (2011)
5. Albert, E., Flores-Montoya, A.E., Genaim, S.: Analysis of May-Happen-in-Parallel in Concurrent Objects. In: Giese, H., Rosu, G. (eds.) FMOODS/FORTE 2012. LNCS, vol. 7273, pp. 35–51. Springer, Heidelberg (2012)
6. Armstrong, J., Virding, R., Wistrom, C., Williams, M.: Concurrent Programming in Erlang. Prentice Hall (1996)
7. Cousot, P., Halbwachs, N.: Automatic Discovery of Linear Restraints Among Variables of a Program. In: POPL. ACM Press (1978)
8. de Boer, F.S., Clarke, D., Johnsen, E.B.: A Complete Guide to the Future. In: De Nicola, R. (ed.) ESOP 2007. LNCS, vol. 4421, pp. 316–330. Springer, Heidelberg (2007)
9. Gulwani, S., Mehra, K.K., Chilimbi, T.M.: Speed: Precise and Efficient Static Estimation of Program Computational Complexity. In: Proc. of POPL 2009, pp. 127–139. ACM (2009)
10. Haller, P., Odersky, M.: Scala actors: Unifying thread-based and event-based programming. Theor. Comput. Sci. 410(2-3), 202–220 (2009)
11. Johnsen, E.B., Hähnle, R., Schäfer, J., Schlatte, R., Steffen, M.: ABS: A Core Language for Abstract Behavioral Specification. In: Aichernig, B.K., de Boer, F.S., Bonsangue, M.M. (eds.) FMCO 2010. LNCS, vol. 6957, pp. 142–164. Springer, Heidelberg (2011)
12. Lee, J.K., Palsberg, J.: Featherweight x10: a core calculus for async-finish parallelism. SIGPLAN Not. 45(5), 25–36 (2010), 1693459
13. Milanova, A., Rountev, A., Ryder, B.G.: Parameterized Object Sensitivity for Points-to Analysis for Java. ACM Trans. Softw. Eng. Methodol. 14, 1–41 (2005)
14. Shapiro, M., Horwitz, S.: Fast and Accurate Flow-Insensitive Points-To Analysis. In: Proc. of POPL 1997, Paris, France, pp. 1–14. ACM (January 1997)
15. Sridharan, M., Bodík, R.: Refinement-based context-sensitive points-to analysis for Java. In: PLDI, pp. 387–400 (2006)
16. Zuleger, F., Gulwani, S., Sinn, M., Veith, H.: Bound analysis of imperative programs with the size-change abstraction. In: Yahav, E. (ed.) SAS 2011. LNCS, vol. 6887, pp. 280–297. Springer, Heidelberg (2011)

Fully Abstract Operation Contracts[⋆]

Richard Bubel, Reiner Hähnle, and Maria Pelevina

Department of Computer Science
Technical University of Darmstadt
{bubel,haehnle}@cs.tu-darmstadt.de, m.pelevina@gmail.com

Abstract. Proof reuse in formal software verification is crucial in presence of constant evolutionary changes to the verification target. Contract-based verification makes it possible to verify large programs, because each method in a program can be verified against its contract separately. A small change to some contract, however, invalidates all proofs that rely on it, which makes reuse difficult. We introduce fully abstract contracts and class invariants which permit to completely decouple reasoning about programs from the applicability check of contracts. We implemented tool support for abstract contracts as part of the KeY verification system and empirically show the considerable reuse potential of our approach.

1 Introduction

A major problem in deductive verification of software is to keep up with changes of the specification and implementation of the target code. Such changes are inevitable, simply because of bug fixes, but also because in industrial practice requirements are constantly evolving. The situation is exacerbated when, as it is often the case, software systems possess a high degree of variability (e.g., in product line-based development).

To redo most proofs and, in the process of doing this, to re-examine most specifications is expensive and slow, even in the ideal case when no user interaction with the verification tool is required. Therefore, a research question in formal verification that is of the utmost practical relevance is how to minimize the amount of verification effort necessary after changes in the underlying specification and implementation of the target code.

Like most state-of-art formal verification approaches, we assume to work in a *contract-based* [1] setting, where the *granularity* of specification units is at the level of one method. The most important advantage of method contracts is the following notion of *compositionality*: assume a program consists of methods $m \in M$, each specified with a contract C_m. Now we prove that each method m satisfies its contract. During this proof, calls inside m to other methods n are handled by appying their contract C_n instead of inlining the code of n. This works fine as long as there are no recursive method calls.[1] This contract-based

⋆ This work has been partially supported by EC FP7 Project No. 610582 ENVISAGE.

[1] There are ways to extend this methodology to recursive method calls, but to keep the presentation simple, we explicitly exclude recursion in this paper. It is an issue that is orthogonal to the techniques discussed here.

T. Margaria and B. Steffen (Eds.): ISoLA 2014, Part II, LNCS 8803, pp. 120–134, 2014.

verification approach is implemented in many state-of-art verification systems, including KeY [2].

In the context of contract-based verification, the problem of keeping up with target code changes can be formulated as follows: assume we have successfully verified a given piece of code p. Now, one of the methods m called in p is changed, i.e., m's contract C_m in general is no longer valid. Therefore, C_m cannot be used in the proof of p which is accordingly broken and must be redone with the new contract of m. If m occurs in many places in p, this becomes very expensive.

In previous work [3] we presented a novel approach to mitigate the problem by introducing the notion of an *abstract contract*. Abstract contracts do not provide concrete pre- or postconditions, but only placeholders for them. At the same time, abstract contracts can be used in a verification system exactly in the same manner as standard concrete contracts. Even though the proofs resulting from abstract contracts can (most of the times) not be closed, we still end up with partial proofs whose open goals are pure first-order. These open proofs (or just the open goals) can be cached, reused, and then verified for different instantiations for the placeholders. In the present paper we make three contributions:

1. A crucial part of each method contract is its *assignable clause*, a list of references to memory locations whose value might be affected by the execution of that method. Obviously, the assignable clause is necessary for sound usage of contracts. In [3] we imposed the restriction that assignable clauses could not be abstract, they had to be explicitly given. This is a serious restriction, because it assumes that the changeable locations of a method cannot increase when its specification is changed. In the present paper we lift this restriction and, therefore, arrive at a notion of *fully abstract* method contracts.
2. The specification of a software system usually consists not only of contracts, but also of class invariants.[2] We extended our approach to cover *abstract invariants*.
3. The evaluation done in [3] was limited by the fact that we had no native implementation of our approach, but had to simulate the effects with suitably hand-crafted inputs. Now we have native tool support, integrated into the KeY verification system for the full Java fragment supported by KeY. Specifically, we evaluate our approach with two case studies. One of them was taken unaltered from a different project [4] to demonstrate that our approach works well in situations not "tailored" to it.

The paper is structured as follows: Section 2 introduces the necessary notions and theoretical background. Section 3 introduces fully abstract contracts and explains their usage. We evaluate our approach with two case-studies in Section 4 and conclude the paper with a discussion about related work (Section 5) and a conclusion (Section 6).

[2] Invariants can be simulated by contracts, but specifications become much more succinct with them, so it is well worth to support them directly.

2 Background

Here we present the background required to understand our specification and verification approach. Our programming language is sequential Java without floats, garbage collection or dynamic class loading. The specification approach follows the well-known design-by-contract paradigm [1].

Example 1. Our running example is a software fragment that computes whether a student passed a course. The concrete passing criteria give rise to different program versions. The basic version, where a student passes when she has passed the exam and all labs, is shown in Fig. 1.

The program consists of a single class `StudentRecord` which implements two methods. Method `passed()` is the main method determining whether the student has passed. It contains a loop to determine whether all labs were finished successfully and it invokes method `computeGrade()` to access the exam grade.

```
class StudentRecord {
  int exam; // exam result
  int passingGrade; // minimum grade necessary to pass the exam
  boolean[] labs = new boolean[10];  // labs

  //@ public invariant exam >= 0 && passingGrade >= 0 && ;
  //@ public invariant labs.length == 10;

  /*@ public normal_behavior
    @ requires true;
    @ ensures \result == exam;
    @ assignable \nothing; @*/
  int computeGrade(){ return exam; }

  /*@ public normal_behavior
    @ ensures \result ==> exam >= passingGrade;
    @ ensures \result ==> (\forall int x; 0 <= x && x < 10; labs[x]);
    @ assignable \nothing; @*/
  boolean passed() {
   boolean enoughPoints = computeGrade() >= passingGrade;
   boolean allLabsDone = true;
   for (int i = 0; i < 10; i++) {
       allLabsDone = allLabsDone && labs[i];
   }
   return enoughPoints && allLabsDone;
  }
}
```

Fig. 1. Implementation and specification of class `StudentRecord` in its basic version

As specification language we use the Java Modeling Language (JML) [5]. JML allows one to specify instance invariants and method contracts. Instance invariants express properties about objects which are established by the constructor and must be preserved throughout the lifetime of an object. More precisely, if an instance invariant holds at the beginning of a method, then it must hold again when it returns, even though it may be violated during its execution.

Example 2. In JML invariants are specified following the keyword **`invariant`** and a boolean expression. The class `StudentRecord` has two invariants on line 6–7. The first specifies lower bounds for the grade of an exam and the minimum grade required to pass the exam. The second invariant fixes the number of labs to ten. In addition, there are implicit invariants, because JML has a "non-null by default" semantics: for each field with a reference type there is an invariant specifying that it is non-null. All invariants are implicitly conjunctively connected into a single instance invariant (per class).

JML method contracts are specified as comments that appear just before the method they relate to. A JML method contract starts with the declaration of a specification case such as **`normal_behavior`** or **`exceptional_behavior`**. For ease of presentation we consider only method contracts specifying the normal behaviour, i.e., normal termination of the method without throwing an uncaught exception. The generalisation to exceptional behaviour specification cases is straightforward. We introduce JML method contracts by way of example:

Example 3. The contract for method `computeGrade()` has one normal behaviour specification case which consists of a precondition (keyword **`requires`**) and a postcondition (keyword **`ensures`**). The actual condition is a boolean expression following the keyword. JML specification expressions are boolean Java expression plus quantifiers and a few extra operators. For instance, in postconditions it is possible to access the return value of the method using the keyword **`\result`** or to refer to the value of an expression `e` in the prestate of the method by **`\old`**`(e)`.

Also part of the specification case definition is the **`assignable`** clause (also called modifies clause) which determines the set of locations that may be changed *at most* by the method to which the contract applies.

In the contract of `computeGrade()` the precondition is simply **`true`** which is the default and can be omitted. By default the invariant for the callee object (i.e., the object referred to by **`this`**) is implicitly added as a precondition. The postcondition states that the return value of the method equals the value of the field containing the exam grade and the assignable clause specifies the empty set of locations which expresses that the method is not allowed to modify any object field or array element. A method may have several specification cases of the same kind with the obvious semantics.

Definition 1 (Program location). *A* program location *is a pair* (o, f) *consisting of an object* o *and a field* f*.*

A program location is an access point to a memory location that a program might change. Next we formalize method contracts and invariants:

Definition 2 (Method contracts, invariants). *Let* `B` *denote a Java class. An* (instance) invariant *for* `B` *is a formula* $inv_{\texttt{B}}(\texttt{v\_this}, \texttt{v\_heap})$ *where* `v_this`*,* `v_heap` *are program variables that refer to the current* **`this`** *object and the heap under which the invariant is evaluated.*

Let T $\texttt{m}(S_1\ a_1, \ldots, S_n\ a_n)$ *be a method with return type* T *and parameters* a_i *of type* S_i*. A* method contract

$$C_{\texttt{m}} = (pre, post, mod)$$

for `m` *consists of*

- *a formula pre specifying the method's precondition;*
- *a formula post specifying the method's postcondition;*
- *an assignable clause mod which is a list of terms specifying the set of program locations that might be changed by method* `m`*.*

Each constituent of a contract may refer to method parameters $\bar{a} = (a_1, \ldots, a_n)$*, the callee using the program variable* `v_this`*, and the current Java heap using the program variable* `v_heap`*. In addition, the postcondition can refer to the result value of a method using the program variable* `result` *as well and to the value of a term/formula in the methods prestate by enclosing it using the transformer function old.*

When verifying whether a method satisfies its contract, we ensure that, if a method is invoked in a state where the invariant of the **`this`** object as well as the method's precondition holds, then in the final state after the method execution, both, its postcondition and the instance invariant are satisfied. Stated as a Hoare triple [6] it looks as follows:

$$\{inv \wedge pre\} \quad \texttt{o.m}(\bar{a}) \quad \{inv \wedge post\}$$

If the method under verification invokes another method `n` of class `B` with contract $(pre, post, mod)$, then, during the deductive verification process, we reach the point where we need to use `n`'s contract and we need to apply the corresponding operation contract rule:

$$\mathsf{useMethodContract}_{\mathrm{C_n}} \quad \frac{\begin{array}{l}\vdash inv(heap, o) \wedge pre(heap, o, \bar{a}) \\ \vdash P(heap) \wedge post(U(heap), heap, o, \bar{a}) \rightarrow Q(U(heap))\end{array}}{\{P(heap)\} \quad o.\texttt{n}(\bar{a}) \quad \{Q(heap)\}} \quad ,$$

where $U(heap) := U(mod)(heap)$ is a transformer that rewrites the heap representation such that all knowledge about the values of the program locations contained in *mod* is removed and everything else is unchanged.

The method contract rule generates two proof obligations: the first ensures that the precondition of `n` and the invariant of o hold. The second checks whether the information in the poststate is sufficient to prove the desired property.

A few words on practical issues concerning the transformer U. Such a function can be realized in different ways, depending on the underlying program logic.

In our implementation we use an explicit heap model which is formalized as a theory of arrays. The transformer U is then realized with the help of a function $anon(h1, mod, h2)$ which is axiomatized to return a heap that coincides with $h1$ for all program locations not in mod and with $h2$ otherwise. The transformer then introduces a new skolem constant h_{sk} of the heap datatype and returns the new heap $anon(heap, mod, h_{sk})$.

3 Abstract Operation Contracts

Abstract operation contracts have been introduced in [3] for the modeling language ABS [7] by some of the co-authors of this paper. The original version required the assignable clause to be concrete. This section presents a fully abstract version of operation contracts for the Java language including an abstract assignable clause and abstract instance invariants. We introduce first the notion of *placeholders* which are declared and used by abstract operation contracts:

Definition 3 (Placeholder). *Let* `B` *be a class and let* T $\mathtt{m}(T_1\ p_1, \ldots, T_n\ p_n)$ *denote a method declared in* `B`. *A* placeholder *is an uninterpreted predicate or function symbol of one of the following four types:*

Requires placeholder *is a predicate* $R(Heap, \mathtt{B}, T_1, \ldots, T_n)$ *which depends on the heap at invocation time, the callee (the object represented by* **`this`** *in* `m`*), and the method arguments.*

Ensures placeholder *is a predicate* $E(Heap, Heap, \mathtt{B}, T, T_1, \ldots, T_n)$ *which depends on the heap in the method's final state, the heap at invocation time (to be able to refer to old values), the callee, the result value of* `m`*, and the method arguments.*

Assignable placeholder *is a function* $A(Heap, \mathtt{B}, T_1, \ldots, T_n)$ *with return type LocSet representing a set of locations; the set is dynamic and may depend on the heap at invocation time, the callee and the method arguments.*

(Instance) invariant placeholder *is a predicate* $I(Heap,\ \mathtt{B})$ *which may depend on the current heap and the instance (the* **`this`** *object) for which the invariant should hold.*

Placeholders allow to delay the actual instantiation of a contract or invariant. We extend the notion of invariant and contract from the previous section by introducing placeholders for the various constituents.

Definition 4 (Extended invariant). *An* extended instance invariant $Inv_{\mathtt{B}} := (I_{\mathtt{B}}, Decls, def_I)$ *of class* `B` *consists of*

- *a unique invariant placeholder* $I_{\mathtt{B}}$ *for class* `B`*;*
- *a list Decls declaring the variables* `v_heap` *of type* `Heap` *and* `v_this` *of type* `B`*;*
- *a formula* def_I *which specifies the semantics of* $I_{\mathtt{B}}$ *and that can make use of the two variables declared in Decls.*

We refer to formula $I_{\mathtt{B}}(\mathtt{v\_heap}, \mathtt{v\_this})$ *as an* abstract invariant.

Definition 5 (Extended operation contract). *An* extended operation contract $C_{\mathtt{m}} = (\mathit{Decls}, C_a, \mathit{defs})_{\mathtt{m}}$ *for a method* `m` *consists of*

- *Decls a list of variable and placeholder declarations;*
- $C_a := (\mathit{pre}_a, \mathit{post}_a, \mathit{mod}_a)$ *the* abstract operation contract *where the different constituents adhere to Def. 6;*
- *defs a list of pairs* (P, def_P) *with a formula* def_P *for each declared placeholder* $P \in \mathit{Decls}$.

Definition 6 (Abstract clauses). *The clauses for* abstract preconditions pre^a, abstract postconditions post^a, *and* abstract assignable clauses mod^a *are defined by the following grammar:*

$$\begin{aligned}
\mathit{pre}^a &::= R \mid I \wedge \mathit{pre}^a \mid \mathit{pre}^a \wedge \mathit{pre}^a \\
\mathit{post}^a &::= E \wedge I \mid E \wedge \mathit{pre}^a \mid I \wedge \mathit{pre}^a \\
\mathit{mod}^a &::= A \mid \mathit{mod}^a \cup \mathit{mod}^a
\end{aligned}$$

where R, E, A *and* I *are atomic formulas with a requires, ensures, assignable or invariant placeholder as top level symbol.*

To render extended invariants and operation contracts within JML we added some keywords which we explain along our running example:

Example 4. The following method contract for `computeGrade()` expresses semantically the same as the one shown in Fig. 1:

```
/*@ public normal_behavior
  @ requires_abs computeGradeR;
  @ ensures_abs  computeGradeE;
  @ assignable_abs computeGradeA;
  @
  @ def computeGradeR = true;
  @ def computeGradeE = \result == exam;
  @ def computeGradeA = \nothing;
  @*/
```

The contract is divided into two sections: the *abstract section* is in the upper section and uses the keywords **`requires_abs`**, **`ensures_abs`**, and **`assignable_abs`**. They are mainly used to declare the placeholders' names. The *concrete section* of the contract consists of the last three lines which provide the definition of the placeholders.

For invariants we do not modify the syntax at all. The placeholder name is generated automatically and the specified **`invariant`** expression is used as its definition. In our logical framework based on KeY, invariants are modelled as JML model fields and the specified invariants have been interpreted as represents clauses. Hence, adding support for abstract invariants was straightforward.

The advantage of having extended invariants and operation contracts is that the abstract contract can be used instead of a concrete contract when applying the operation contract rule. It is also not necessary to modify the operation contract rule which guarantees the soundness of our approach as long as the original calculus was sound.

By using only abstract contracts, we can construct a proof for the correctness of a method `m` without committing to any concrete instantiation of an abstract contract neither for `m` itself nor for any of the methods it invokes. Once the verification conditions for the whole program are computed (e.g., by symbolic execution, constraint propagation, etc.), possibly followed by additional simplification steps, the open proof (goals) can be saved and cached.

To support abstract assignable clauses the underlying calculus must provide means to represent the modified locations within its logic. As mentioned in the previous section, our formalisation uses the explicit function *anon* which takes a location set as argument. Instead of providing a concrete set we use simply its placeholder.

To finish the proof that a method `m` adheres to its concrete specification we can reuse the cached proof (goals) without the need to redo the program transformation and simplification steps (as long as the implementation of `m` has not changed). It suffices to replace the introduced placeholders by their actual definitions and then continue proof search as usual. Replacing the placeholders by their definitions can be achieved, for example, by translating each placeholder definition into a rewrite rule of the underlying calculus which is added as an axiom.

The saving potential of our approach is particularly strong when verifying software with a high variability. In addition, our approach does not necessarily require to adhere to the Liskov principle [8] for specification but allows for a more flexible approach. We discuss the advantages in detail in the following Section.

4 Evaluation

We evaluate empirically the benefits offered by abstract contracts by measuring the amount of possible proof reuse. The evaluation is based on an implementation of our ideas in KeY. Our conjecture is that using abstract contracts results in considerable savings in terms of proof effort.

We evaluated our approach using two case studies of which each is implemented in several variants. The different variants have a common top-level interface, but differ in the implementation as well as in the concrete specification of called sub-routines. The abstract specification of each sub-routine, however, stays the same.

4.1 Description

The first case study *Student Record* has been already introduced before as a running example. We implemented several variants which differ in the implementation and specification of method `computeGrade()`, but keep method `passed()`

unchanged. The three variants differ in whether and how bonus credits are considered when computing the final exam grade. Version 3 is the version which supports no bonus points at all.

The second case study is taken from [4] and implements two classes, `Account` and `Transaction`. It is implemented in five versions. The most basic variant is shown in (Fig. 2). The class `Account` implements a simple bank account with method **`boolean`** `update(`**`int`** `x)` for deposit/withdrawal operations (depending on the sign of parameter `x`). In addition, there is an inverse operation **`boolean`** `undoUpdate(`**`int`** `x)`. Both methods signal whether the operation was successful with their return value. In the basic variant these methods perform the update unconditionally (for instance, it is not checked whether the account has enough savings).

The bank account can also be locked to prevent the execution of any operation. Locking is supported by methods `lock()`, `unLock()` and `isLocked()`. The behavior of these methods is not varied among the different variants.

Transactions between accounts are performed by calls to method `transfer` implemented by class `Transaction`. Method `transfer()` performs a number of pre-checks to ensure that the transaction will be successful before actually performing the transfer. For instance, it is checked that the balance of the source account is not negative, none of the accounts is locked, etc.

The versions differ in specification and/or implementation. The second version keeps the implementation of all classes unchanged, but simplifies the contract of method `transfer()` to cover only the case where a transaction is unsuccessful because of a negative amount to be transferred. In the third version an overdraft limit for accounts is supported, i.e., accounts are only allowed to be in the negative until a certain limit. This feature requires changes in the implementation and specification of methods `update()` and `undoUpdate()`. Method `transfer()` needs not to be modified with respect to the basic version. The fourth version extends the third version by adding a daily withdrawal limit. Again changes are necessary for the methods `update()` and `undoUpdate()`. The fifth version is again an extension of the third adding a fee to account operations. The changes affect the implementation and specification for the methods `update()`, `undoUpdate()` and the specification of method `transfer()`.

4.2 Results

For each program in the case studies we conducted three experiments with (i) fully abstract contracts, (ii) partially abstract contracts (the assignable clause is concrete, this correpsonds to the setup in [3]), and (iii) concrete contracts.

In each run where the specification was fully or partially abstract, the first step was to construct a partial proof that eliminates the verified programs and only uses abstract contracts and abstract invariants. This partial proof was then cached and reused to actually verify that the different variants satisfy their specifications. In each experiment with concrete contracts only, we directly ran the verification process on each program version.

```
public class Account {
  int balance = 0;
  boolean lock = false;

 /*@ public normal_behavior
   @ ensures (balance == \old(balance) + x) && \result;
   @ assignable balance;
   @*/
  boolean update(int x) { balance = balance + x; return true; }

 /*@ public normal_behavior
   @ ensures (balance == \old(balance) - x) && \result;
   @ assignable balance;
   @*/
  boolean undoUpdate(int x) { balance = balance - x; return true; }

 /*@ public normal_behavior
   @ ensures \result == this.lock;
   @*/
  boolean /*@ pure @*/ isLocked() { return lock; }
}

public class Transaction {
 /*@ public normal_behavior
   @ requires dest != null && source != null && source != destination;
   @ ensures \result ==> (\old(dest.balance) + amount == dest.balance);
   @ ensures \result ==> (\old(source.balance) - amount == source.balance);
   @ assignable \everything;
   @*/
  public boolean transfer(Account source, Account dest, int amount) {
   if (source.balance < 0) amount = -1;
   if (dest.isLocked()) amount = -1;
   if (source.isLocked()) amount = -1;
   int take, give;
   if (amount != -1) { take = amount * -1; give = amount; }
   if (amount <= 0) { return false; }
   if (!source.update(take)) { return false; }
   if (!dest.update(give)) {
    source.undoUpdate(take);
    return false;
   }
   return true;
}}
```

Fig. 2. Implementation and specification of the classes `Account` and `Transfer` (Ver. 1)

We measured the complexity of the cached proof and the final proofs in terms of nodes and branches (given in parenthesis). The results for the case studies are shown in Table 1 and 2. The column *Savings* shows the amount of proof effort that has been saved (in terms of nodes) by reusing the cached proof. The row *Total* shows the aggregated proof effort (in nodes) for all program versions. For the experiments using concrete contracts the total equals the sum of the number of nodes for the different versions, i.e., $total = \sum_{i=1}^{n} \#nodes_{\mathrm{v}i}$

Table 1. Results for `StudentRecord` example

Version	Completely abstract			Partially Abstract			Concrete
	Partial proof	Full proof	Savings	Partial proof	Full proof	Savings	Full proof
v1	514 (9)	1191 (25)	43%	519 (9)	1145 (25)	45%	919 (21)
v2		1806 (40)	28%		1782 (40)	29%	1419 (36)
v3		1009 (24)	51%		937 (24)	55%	904 (22)
Total	2978			2826			3242

with n being the number of program versions. In case of the experiments using partially or completely abstract contracts, we compute the total as for concrete contracts, but subtract $n-1$ times the size of the partial proof, i.e., $total = (\sum_{i=1}^{n} \#nodes_{\mathrm{v}i}) - (n-1) \cdot nodes_{partial}$, to account for the proof reuse.

Table 2. Results for `Account` example

Version	Completely abstract			Partially Abstract			Concrete
	Partial proof	Full proof	Savings	Partial proof	Full proof	Savings	Full proof
v1	1390 (55)	2066 (57)	67%	1408 (55)	1865 (55)	75%	1035 (63)
v2		1882 (57)	74%		1723 (55)	82%	972 (63)
v3		2435 (59)	57%		1896 (55)	74%	1352 (68)
v4		2719 (59)	51%		1975 (55)	71%	1461 (68)
v5		2701 (61)	51%		2073 (57)	68%	1601 (69)
Total	6243			3900			6421

The results show that we achieve a reduced overall effort for fully abstract contracts as well as for partially abstract contracts. For the considered examples, the savings are most significant for partially abstract contracts. The reason is that here assignable clause is concrete, so the application of the contract rule keeps more information about the heap, while in the fully abstract case almost all information about the heap is wiped out. As a consequence, branches checking for `NullPointerExceptions`, etc., cannot be closed in the abstract proofs.

Another reason why fully abstract contracts save perhpas less than expected is that in the case studies the assignable clauses in the concrete case contain only few locations and are mostly stable among the different versions. In the final version of this paper we will include a case study that shows more variability

Table 3. Analysis of the `StudentRecord` example (specified with concrete contracts)

Version	Symbolic execution	Full proof	Ratio
v1	442 (11)	919 (21)	48%
v2	494 (11)	1419 (36)	35%
v3	410 (11)	904 (22)	45%

with respect to the set of assignable locations to illustrate the savings potential of fully abstract contracts.

Tables 3 and 4 show the ratio of proof nodes concerned with symbolic execution (i.e., program transformation) as compared to first-order reasoning. It can be directly seen that this ratio and the observed savings are strongly correlated.

Table 4. Analysis of the `Account` example (specified with concrete contracts)

Version	Symbolic execution	Full proof	Ratio
v1	842 (53)	1035 (63)	81%
v2	818 (53)	972 (63)	84%
v3	1037 (56)	1352 (68)	77%
v4	1147 (56)	1461 (68)	79%
v5	1141 (56)	1601 (69)	71%

Fig. 3 illustrates the *amortized* proof complexity of the `Account` example: the proof effort spent for the partial proof is distributed uniformly among the proofs of all five program versions (in case of partial and fully abstract contracts, respectively). We can see that using partial contracts perform best, while using fully abstract contracts leads in most cases to a total proof effort comparable to concrete ones. The reason is that when using abstract assignable clauses most knowledge about the heap after a method invocation is lost, prohibiting certain simplifications and creating more complex first-order problems. Currently, we investigate whether using SMT solvers for first-order goals (instead of the built-in KeY prover) can help. This should definitely be the case, provided that by using SMT solvers one can achieve a considerable speed-up for first-order inference.

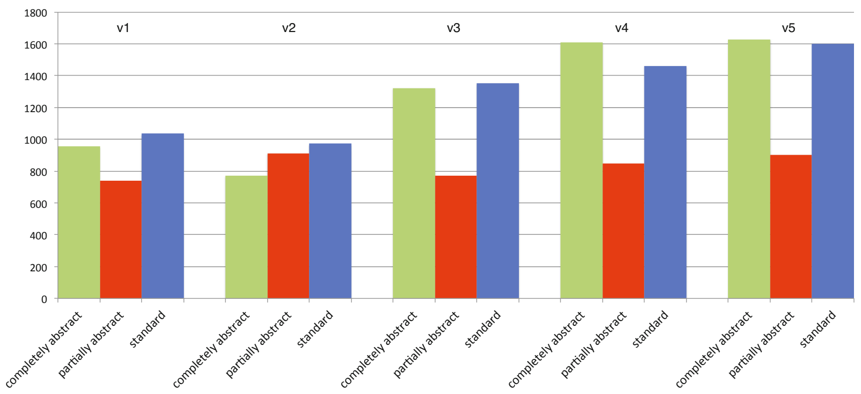


Fig. 3. Amortized proof complexity (in nodes) for the `Account` example

5 Related Work

As our work builds upon previous work [3] of some of the co-authors this is also the closest related work. As stated in the introduction, in the present paper we added abstract assignable clauses, abstract invariants, a proper implementation in KeY, plus an evaluation.

Proof reuse has been studied, for instance, in [9,10] where proof replay is proposed to reduce the verification effort. The old proof is replayed and when this is no longer possible, a new proof rule is chosen heuristically: in [9] a similarity measure is utilized, while in [10] differencing operations are applied. The proof reuse focusses only on the proof structure and does not take the specification into account like our work.

In [11], a set of allowed changes to evolve an OO program is introduced. For verified method contracts, a proof context is constructed which keeps track of proof obligations. Program changes cause the proof context to be adapted so that the proof obligations that are still valid are preserved and new proof goals are created. Earlier work along the same lines in the context of VCG is [12].

Reasoning by analogy is applied in [13] to reuse problem solving experience in proof planning. Generalization of proofs [14,15] facilitates to reuse proofs in different contexts.

For formal software development in the large [16], evolving formal specifications are maintained by representing the dependencies between formal specifications and proofs in a development graph. For each modification, the effect in the development graph is computed such that only invalidated proofs have to be re-done. In [17], proofs are evolved together with formal specifications. A set of basic transformation operations for specifications induces the corresponding transformations of the proofs which may include the creation of new proof obligations.

In [18] it is assumed that one program variant has been fully verified. By analyzing the differences to another program variant one obtains those proof obligations that remain valid in the new product variant and that need not be reestablished. In [19] methods are verified based on a contract which makes assumptions on the contracts of the called methods explicit.

In [20] abstract predicates are defined as abbreviations for specific properties to enforce modular reasoning in the sense of information hiding. Inside a module the definition of the abstract properties can be expanded while outside a module only the abstract predicates can be used. The paper does not use these abstract predicates to decouple symbolic execution from the application check of contracts and an application to proof reuse through caching is not considered.

We are not aware of any specification approach that keeps the used contracts fully abstract by using placeholder functions and predicates, as done here.

6 Conclusion and Future Work

We introduced fully abstract method contracts and class invariants in the context of contract-based verification. We showed that this idea can be simply realized with the help of placeholders. We do not assume any specific specification language, target language, or program logic. Any sufficiently expressive program logic can be used for an implementation. We instantiated and implemented the framework as part of the KeY verification system with Java as target language, JML as specification language, and dynamic logic as the program logic. An experimental evaluation showed that considerable proof reuse in the presence of changes and variations to specifications and code is possible.

In future work, we intend to investigate amount of savings that is achievable when using *mixed* contracts, where abstract contracts are enriched with certain concrete expressions that can be assumed to be stable across different versions of a program. Examples are formal parameters or field values can never be null, system values and boundaries, etc. Using this kind of information should allow us to simplify the cached proofs even further and increase the savings ratio. Further, we will investigate whether an inverse assignable clause (which lists the fields a method must *not* change) makes sense. Such a clause would enable us to retain more information after a method invocation and allow for a better reuse potential.

References

1. Meyer, B.: Applying “design by contract”. IEEE Computer 25(10), 40–51 (1992)
2. Beckert, B., Hähnle, R., Schmitt, P.H. (eds.): Verification of Object-Oriented Software. LNCS (LNAI), vol. 4334. Springer, Heidelberg (2007)
3. Hähnle, R., Schaefer, I., Bubel, R.: Reuse in software verification by abstract method calls. In: Bonacina, M.P. (ed.) CADE 2013. LNCS, vol. 7898, pp. 300–314. Springer, Heidelberg (2013)
4. Thüm, T., Schaefer, I., Apel, S., Hentschel, M.: Family-based deductive verification of software product lines. In: Proceedings of the 11th International Conference on Generative Programming and Component Engineering, GPCE 2012, pp. 11–20. ACM, New York (2012)

5. Leavens, G.T., Poll, E., Clifton, C., Cheon, Y., Ruby, C., Cok, D., Müller, P., Kiniry, J., Chalin, P., Zimmerman, D.M.: JML Reference Manual. Draft revision 1.235 (September 2009)
6. Hoare, C.A.R.: An axiomatic basis for computer programming. Communications of the ACM 12(10) (October 1969)
7. Johnsen, E.B., Hähnle, R., Schäfer, J., Schlatte, R., Steffen, M.: ABS: A core language for abstract behavioral specification. In: Aichernig, B.K., de Boer, F.S., Bonsangue, M.M. (eds.) FMCO 2010. LNCS, vol. 6957, pp. 142–164. Springer, Heidelberg (2011)
8. Liskov, B., Wing, J.M.: A behavioral notion of subtyping. ACM Trans. Program. Lang. Syst. 16(6), 1811–1841 (1994)
9. Beckert, B., Klebanov, V.: Proof reuse for deductive program verification. In: Third IEEE International Conference on Software Engineering and Formal Methods, pp. 77–86. IEEE Computer Society (2004)
10. Reif, W., Stenzel, K.: Reuse of proofs in software verification. In: Shyamasundar, R.K. (ed.) FSTTCS 1993. LNCS, vol. 761, pp. 284–293. Springer, Heidelberg (1993)
11. Dovland, J., Johnsen, E.B., Yu, I.C.: Tracking behavioral constraints during object-oriented software evolution. In: Margaria, T., Steffen, B. (eds.) ISoLA 2012, Part I. LNCS, vol. 7609, pp. 253–268. Springer, Heidelberg (2012)
12. Grigore, R., Moskal, M.: Edit & verify. In: First-order Theorem Proving Workshop, Liverpool, UK (2007)
13. Melis, E., Whittle, J.: Analogy in inductive theorem proving. J. Autom. Reasoning 22(2), 117–147 (1999)
14. Walther, C., Kolbe, T.: Proving theorems by reuse. Artificial Intelligence 116(1-2), 17–66 (2000)
15. Felty, A.P., Howe, D.J.: Generalization and reuse of tactic proofs. In: Pfenning, F. (ed.) LPAR 1994. LNCS, vol. 822, pp. 1–15. Springer, Heidelberg (1994)
16. Hutter, D., Autexier, S.: Formal Software Development in MAYA. In: Hutter, D., Stephan, W. (eds.) Mechanizing Mathematical Reasoning. LNCS (LNAI), vol. 2605, pp. 407–432. Springer, Heidelberg (2005)
17. Schairer, A., Hutter, D.: Proof transformations for evolutionary formal software development. In: Kirchner, H., Ringeissen, C. (eds.) AMAST 2002. LNCS, vol. 2422, pp. 441–456. Springer, Heidelberg (2002)
18. Bruns, D., Klebanov, V., Schaefer, I.: Verification of software product lines with delta-oriented slicing. In: Beckert, B., Marché, C. (eds.) FoVeOOS 2010. LNCS, vol. 6528, pp. 61–75. Springer, Heidelberg (2011)
19. Damiani, F., Owe, O., Dovland, J., Schaefer, I., Johnsen, E.B., Yu, I.C.: A transformational proof system for delta-oriented programming. In: SPLC (2), pp. 53–60 (2012)
20. Parkinson, M., Bierman, G.: Separation logic and abstraction. In: Proceedings of the 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2005, pp. 247–258. ACM, New York (2005)

Statistical Model Checking Past, Present, and Future (Track Introduction)

Kim G. Larsen and Axel Legay

[1] Aalborg University, Denmark
[2] INRIA Rennes – Bretagne Atlantique, France

Abstract. This short note introduces statistical model checking and gives a brief overview of the *Statistical Model Checking, past present and future* session at Isola 2014.

1 Context

Quantitative properties of stochastic systems are usually specified in logics that allow one to compare the measure of executions satisfying certain temporal properties with thresholds. The model checking problem for stochastic systems with respect to such logics is typically solved by a numerical approach [BHHK03, CG04] that iteratively computes (or approximates) the exact measure of paths satisfying relevant subformulas; the algorithms themselves depend on the class of systems being analysed as well as the logic used for specifying the properties.

Another approach to solve the model checking problem is to *simulate* the system for finitely many runs, and use *hypothesis testing* to infer whether the samples provide *statistical* evidence for the satisfaction or violation of the specification. This approach was first applied in [LS91], where it was shown that hypothesis testing could be used to settle probabilistic modal logic properties with arbitrary precision, leading in the limit to probabilistic bisimulation. More recently [You05a] this approach has been known as statistical model checking (SMC) and is based on the notion that since sample runs of a stochastic system are drawn according to the distribution defined by the system, they can be used to obtain estimates of the probability measure on executions. Starting from time-bounded PCTL properties [You05a], the technique has been extended to handle properties with unbounded until operators [SVA05b], as well as to black-box systems [SVA04, You05a]. Tools, based on this idea have been built [HLMP04, SVA05a, You05a, You05b, BDD$^+$11, DLL$^+$11, BCLS13], and have been used to analyse many systems that are intractable numerical approaches.

The SMC approach enjoys many advantages. First, the algorithms require only that the system be simulatable (or rather, sample executions be drawn according to the measure space defined by the system). Thus, it can be applied to larger class of systems than numerical model checking algorithms, including black-box systems and infinite state systems. In particular, SMC avoids the ‘state explosion problem’ [CES09]. Second the approach can be generalized to a larger class of properties, including Fourier transform based logics. Third, SMC requires many independent simulation runs, making it easy to parallelise and scale to industrial-sized systems.

While it offers solutions to some intractable numerical model checking problems, SMC also introduces some additional problems. First, SMC only provides probabilistic guarantees about the correctness of the results. Second, the required sample size grows quadratically with respect to the required confidence of the result. This makes rare properties difficult to verify. Third, only the simulation of purely probabilistic systems is well defined. Nondeterministic systems, which are common in the field of formal verification, are especially challenging for SMC.

2 On Statistical Model Checking

Consider a stochastic system $\mathcal{S}$ and a logical property φ that can be checked on finite executions of the system. Statistical Model Checking (SMC) refers to a series of simulation-based techniques that can be used to answer two questions: (1) *Qualitative*: Is the probability for $\mathcal{S}$ to satisfy φ greater or equal to a certain threshold? and (2) *Quantitative*: What is the probability for $\mathcal{S}$ to satisfy φ? In contrast to numerical approaches, the answer is given up to some correctness precision.

In the sequel, we overview two SMC techniques. Let B_i be a discrete random variable with a Bernoulli distribution of parameter p. Such a variable can only take 2 values 0 and 1 with $Pr[B_i = 1] = p$ and $Pr[B_i = 0] = 1 - p$. In our context, each variable B_i is associated with one simulation of the system. The outcome for B_i, denoted b_i, is 1 if the simulation satisfies φ and 0 otherwise.

Qualitative Answer. The main approaches [You05a, SVA04] proposed to answer the qualitative question are based on *sequential hypothesis testing* [Wal45]. Let $p = Pr(\varphi)$. To determine whether $p \geq \theta$, we can test $H : p \geq \theta$ against $K : p < \theta$. A test-based solution does not guarantee a correct result but it is possible to bound the probability of error. The *strength* of a test is determined by two parameters, α and β, such that the probability of accepting K (respectively, H) when H (respectively, K) holds, called a Type-I error (respectively, a Type-II error) is less or equal to α (respectively, β). A test has *ideal performance* if the probability of the Type-I error (respectively, Type-II error) is exactly α (respectively, β). However, these requirements make it impossible to ensure a low probability for both types of errors simultaneously (see [Wal45, You05a] for details). A solution is to use an *indifference region* $[p_1, p_0]$ (given some δ, $p_1 = \theta - \delta$ and $p_0 = \theta + \delta$) and to test $H_0 : p \geq p_0$ against $H_1 : p \leq p_1$. We now sketch the Sequential Probability Ratio Test (SPRT). In this algorithm, one has to choose two values A and B ($A > B$) that ensure that the strength of the test is respected. Let m be the number of observations that have been made so far. The test is based on the following quotient:

$$\frac{p_{1m}}{p_{0m}} = \prod_{i=1}^{m} \frac{Pr(B_i = b_i \mid p = p_1)}{Pr(B_i = b_i \mid p = p_0)} = \frac{p_1^{d_m}(1-p_1)^{m-d_m}}{p_0^{d_m}(1-p_0)^{m-d_m}},$$

where $d_m = \sum_{i=1}^{m} b_i$. The idea is to accept H_0 if $\frac{p_{1m}}{p_{0m}} \geq A$, and H_1 if $\frac{p_{1m}}{p_{0m}} \leq B$. The algorithm computes $\frac{p_{1m}}{p_{0m}}$ for successive values of m until either H_0 or H_1 is satisfied. This has the advantage of minimizing the number of simulations required to make the decision.

Quantitative Answer. In [HLMP04] Peyronnet et al. propose an estimation procedure to compute the probability p for $\mathcal{S}$ to satisfy φ. Given a *precision* δ, the *Chernoff bound* of [Oka59] is used to compute a value for p' such that $|p'-p|\leq\delta$ with *confidence* $1-\alpha$. Let $B_1 \ldots B_m$ be m Bernoulli random variables with parameter p, associated to m simulations of the system considering φ. Let $p' = \sum_{i=1}^{m} b_i/m$, then the Chernoff bound [Oka59] gives $Pr(|p'-p| \geq \delta) \leq 2e^{-2m\delta^2}$. As a consequence, if we take $m = \lceil \ln(2/\alpha)/(2\delta^2) \rceil$, then $Pr(|p'-p|\leq\delta) \geq 1-\alpha$.

2.1 Rare Events

Statistical model checking avoids the exponential growth of states associated with probabilistic model checking by estimating probabilities from multiple executions of a system and by giving results within confidence bounds. Rare properties are often important but pose a particular challenge for simulation-based approaches, hence a key objective for SMC is to reduce the number and length of simulations necessary to produce a result with a given level of confidence. In the literature, one finds two techniques to cope with rare events: *importance sampling* and *importance splitting.*

In order to minimize the number of simulations, importance sampling (see e.g., [Rid10, DBNR00]) works by estimating a probability using weighted simulations that favour the rare property, then compensating for the weights. For importance sampling to be efficient, it is thus crucial to find good importance sampling distributions without considering the entire state space. In [CZ11] Zuliani and Clarke outlined the challenges for SMC and rare-events. A first theory contribution was then provided by Barbot et al. who proposed to use reduction techniques together with cross-entropy [BHP12]. In [JLS12], we presented a simple algorithm that uses the notion of cross-entropy minimisation to find an optimal importance sampling distribution. In contrast to previous work, our algorithm uses a naturally defined low dimensional vector of parameters to specify this distribution and thus avoids the intractable explicit representation of a transition matrix. We show that our parametrisation leads to a unique optimum and can produce many orders of magnitude improvement in simulation efficiency.

One of the open challenges with importance sampling is that the variance of the estimator cannot be usefully bounded with only the knowledge gained from simulation. Importance *splitting* (see e.g., [CG07]) achieves this objective by estimating a sequence of conditional probabilities, whose product is the required result. In [JLS13] motivated the use of importance splitting for statistical model checking and were the first to link this standard variance reduction technique [KM53] with temporal logical. In particular, they showed how to create *score functions* based on logical properties, and thus define a set of *levels* that delimit the conditional probabilities. In [JLS13] they also described the necessary and desirable properties of score functions and levels, and gave two importance splitting algorithms: one that uses fixed levels and one that discovers optimal levels adaptively.

2.2 Nondeterminism

Markov decision processes (MDP) and other nondeterministic models interleave nondeterministic *actions* and probabilistic transitions, possibly with rewards or costs assigned to the actions [Bel57, Put94]. These models have proved useful in many real optimisation problems (see [Whi85, Whi88, Whi93] for a survey of applications of MDPs) and are also used in a more abstract sense to represent concurrent probabilistic systems (e.g., [BDA95]). Such systems comprise probabilistic subsystems whose transitions depend on the states of the other subsystems, while the order in which concurrently enabled transitions execute is nondeterministic. This order may radically affect the expected reward or the probability that a system will satisfy a given property. Numerical model checking may be used to calculate the upper and lower bounds of these quantities, but a simulation semantics is not immediate for nondeterministic systems and SMC is therefore challenging.

SMC cannot be applied to nondeterministic systems without first resolving the nondeterminism using a *scheduler* (alternatively a *strategy* or a *policy*). Since nondeterministic and probabilistic choices are interleaved, schedulers are typically of the same order of complexity as the system as a whole and may be infinite.

In [LS14] Jegouret et al presented the basis of the first lightweight SMC algorithms for MDPs and other nondeterministic models, using an $\mathcal{O}(1)$ representation of history-dependent schedulers. This solution is based on pseudo-random number generators and an efficient hash function, allowing schedulers to be sampled using Monte Carlo techniques. Some previous attempts to apply SMC to nondeterministic models [BFHH11, LP12, HMZ$^+$12, HT13] have been memory-intensive (heavyweight) and incomplete in various ways. The algorithms of [BFHH11, HT13] consider only systems with ‘spurious’ nondeterminism that does not actually affect the probability of a property. In [LP12] the authors consider only memoryless schedulers and do not consider the standard notion of optimality used in model checking (i.e., with respect to probability). The algorithm of [HMZ$^+$12] addresses a standard qualitative probabilistic model checking problem, but is limited to memoryless schedulers that must fit into memory and does not in general converge to the optimal scheduler. Most recently[DJL$^+$], SMC – or reinforcement learning – has been applied to learn near-cost-optimal strategies for priced timed stochastic games subject to guaranteed worst-case time bounds. The method is implemented using a combination of Uppaal-TIGA (for timed games) and Uppaal SMC and provides three alternatives light-weight datastructures for representing stochastic strategies.

3 Content of the Session

SMC has been implemented in several prototypes and tools, which includes Uppaal SMC [DLL+11], Plasma [BCLS13], Ymer [You05b], or COSMOS [BDD+11]. Those tools have been applied to several complex problems coming from a wide range of areas. This includes systems biology (see e.g., [Zul14]), automotive and avionics (see e.g., [BBB+12]), energy-centric systems(see e.g., [DDL+13]), or power grids(see e.g., [HH13]).

This isola session discusses several aspects of SMC, which includes: non-determinism, rare-events, applications to biology/energy-centric/power grids, and runtime verification procedures suited for SMC. Summary of the contributions:

- In [JLS14], the authors propose new SMC techniques for rare-events. The main contribution is in extending [JLS13] with new adaptive level algorithms based on branching simulation, and to show that this permits to get a more precise estimate of the rare probability. In the authors stud
- In [BLT14], the authors propose to apply simulation-based techniques to systems whose number of configurations can vary at execution. They develop a new logic and new SMC techniques for such systems.
- In [BNB+14], the authors provide a complete and detailed comparison of several SMC model checkers, especially for the real-time setting. The authors present five semantic caveats and give a classification scheme for SMC algorithms. They also argue that caution is needed and believe that the caveats and classification scheme in this paper serve as a guiding reference for thoroughly understanding them.
- In [BGGM14], the authors propose an application of SMC to systems biology. More precisely, they consider the Wnt-beta-catenin signaling pathway that plays an important role in the proliferation of neural cells. They analyze the dynamics of the system by combining SMC with the Hybrid Automata Stochastic Logic. in
- In [WHL14], the authors consider wireless systems such as satellites and sensor networks are often battery-powered. The main contributions are (1) to show how SMC can be used to calculate an upper bound on the attainable number of task instances from a battery, and (2) to synthesize a battery-aware scheduler that wastes no energy on instances that are not guaranteed to make their deadlines.
- In [GPS+14], the authors consider CTL-based measuring on paths, and generalize the mea- surement results to the full structure using optimal Monte Carlo estimation techniques. To experimentally validate their framework, they present LTL-based measurements for a flocking model of bird-like agents.
- In [EHFJ+14], the authors explore the effectiveness and challenges of using monitoring techniques, based on Aspect-Oriented Programming, to block adware at the library level, on mobile devices based on Android using miAdBlocker. The authors also present the lessons learned from this experience,

and we identify some challenges when applying runtime monitoring techniques to real-world case studies.
- In [Hav14], the author presents a form of automaton, referred to as data automata, suited for monitoring sequences of data-carrying events, for example emitted by an executing software system. He also presents and evaluate an optimized external DSL for data automata, as well as a comparable unoptimized internal DSL (API) in the Scala programming language, in order to compare the two solutions.

References

[BBB+12] Basu, A., Bensalem, S., Bozga, M., Delahaye, B., Legay, A.: Statistical abstraction and model-checking of large heterogeneous systems. STTT 14(1), 53–72 (2012)

[BCLS13] Boyer, B., Corre, K., Legay, A., Sedwards, S.: PLASMA-lab: A flexible, distributable statistical model checking library. In: Joshi, K., Siegle, M., Stoelinga, M., D'Argenio, P.R. (eds.) QEST 2013. LNCS, vol. 8054, pp. 160–164. Springer, Heidelberg (2013)

[BDA95] Bianco, A., De Alfaro, L.: Model checking of probabilistic and nondeterministic systems. In: Thiagarajan, P.S. (ed.) FSTTCS 1995. LNCS, vol. 1026, pp. 499–513. Springer, Heidelberg (1995)

[BDD+11] Ballarini, P., Djafri, H., Duflot, M., Haddad, S., Pekergin, N.: Cosmos: A statistical model checker for the hybrid automata stochastic logic. In: QEST, pp. 143–144. IEEE Computer Society (2011)

[Bel57] Bellman, R.: Dynamic Programming. Princeton University Press (1957)

[BFHH11] Bogdoll, J., Ferrer Fioriti, L.M., Hartmanns, A., Hermanns, H.: Partial order methods for statistical model checking and simulation. In: Bruni, R., Dingel, J. (eds.) FMOODS/FORTE 2011. LNCS, vol. 6722, pp. 59–74. Springer, Heidelberg (2011)

[BGGM14] Ballarini, P., Gallet, E., Le Gall, P., Manceny, M.: Formal analysis of the wnt/β-catenin pathway through statistical model checking. In: Margaria, T., Steffen, B. (eds.) ISoLA 2014, Part II. LNCS, vol. 8803, pp. 193–207. Springer, Heidelberg (2014)

[BHHK03] Baier, C., Haverkort, B.R., Hermanns, H., Katoen, J.-P.: Model-checking algorithms for continuous-time markov chains. IEEE 29(6), 524–541 (2003)

[BHP12] Barbot, B., Haddad, S., Picaronny, C.: Coupling and importance sampling for statistical model checking. In: Flanagan, C., König, B. (eds.) TACAS 2012. LNCS, vol. 7214, pp. 331–346. Springer, Heidelberg (2012)

[BLT14] Boyer, B., Legay, A., Traonouez, L.-M.: A formalism for stochastic adaptive systems. In: Margaria, T., Steffen, B. (eds.) ISoLA 2014, Part II. LNCS, vol. 8803, pp. 160–176. Springer, Heidelberg (2014)

[BNB+14] Bohlender, D., Bruintjes, H., Junges, S., Katelaan, J., Nguyen, V.Y., Noll, T.: A review of statistical model checking pitfalls on real-time stochastic models. In: Margaria, T., Steffen, B. (eds.) ISoLA 2014, Part II. LNCS, vol. 8803, pp. 177–192. Springer, Heidelberg (2014)

[CES09] Clarke, E.M., Emerson, E.A., Sifakis, J.: Model checking: algorithmic verification and debugging. Commun. ACM 52(11), 74–84 (2009)

[CG04] Ciesinski, F., Größer, M.: On probabilistic computation tree logic. In: Baier, C., Haverkort, B.R., Hermanns, H., Katoen, J.-P., Siegle, M. (eds.) Validation of Stochastic Systems. LNCS, vol. 2925, pp. 147–188. Springer, Heidelberg (2004)

[CG07] Cérou, F., Guyader, A.: Adaptive multilevel splitting for rare event analysis. Stochastic Analysis and Applications 25, 417–443 (2007)

[CZ11] Clarke, E.M., Zuliani, P.: Statistical model checking for cyber-physical systems. In: Bultan, T., Hsiung, P.-A. (eds.) ATVA 2011. LNCS, vol. 6996, pp. 1–12. Springer, Heidelberg (2011)

[DBNR00] De Boer, P.-T., Nicola, V.F., Rubinstein, R.Y.: Adaptive importance sampling simulation of queueing networks. In: Winter Simulation Conference, vol. 1, pp. 646–655 (2000)

[DDL$^+$13] David, A., Du, D., Guldstrand Larsen, K., Legay, A., Mikučionis, M.: Optimizing control strategy using statistical model checking. In: Brat, G., Rungta, N., Venet, A. (eds.) NFM 2013. LNCS, vol. 7871, pp. 352–367. Springer, Heidelberg (2013)

[DJL$^+$] David, A., Jensen, P.G., Larsen, K.G., Legay, A., Lime, D., Sorensen, M.G., Taankvist, J.H.

[DLL$^+$11] David, A., Larsen, K.G., Legay, A., Mikučionis, M., Wang, Z.: Time for statistical model checking of real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 349–355. Springer, Heidelberg (2011)

[EHFJ$^+$14] El-Harake, K., Falcone, Y., Jerad, W., Langet, M., Mamlouk, M.: Blocking advertisements on android devices using monitoring techniques. In: Margaria, T., Steffen, B. (eds.) ISoLA 2014, Part II. LNCS, vol. 8803, pp. 239–253. Springer, Heidelberg (2014)

[GPS$^+$14] Grosu, R., Peled, D., Ramakrishnan, C.R., Smolka, S.A., Stoller, S.D., Yang, J.: Using statistical model checking for measuring systems. In: Margaria, T., Steffen, B. (eds.) ISoLA 2014, Part II. LNCS, vol. 8803, pp. 223–238. Springer, Heidelberg (2014)

[Hav14] Havelund, K.: Monitoring with data automata. In: Margaria, T., Steffen, B. (eds.) ISoLA 2014, Part II. LNCS, vol. 8803, pp. 254–273. Springer, Heidelberg (2014)

[HH13] Hermanns, H., Hartmanns, A.: An internet inspired approach to power grid stability. it - Information Technology 55(2), 45–51 (2013)

[HLMP04] Hérault, T., Lassaigne, R., Magniette, F., Peyronnet, S.: Approximate probabilistic model checking. In: Steffen, B., Levi, G. (eds.) VMCAI 2004. LNCS, vol. 2937, pp. 73–84. Springer, Heidelberg (2004)

[HMZ$^+$12] Henriques, D., Martins, J.G., Zuliani, P., Platzer, A., Clarke, E.M.: Statistical model checking for Markov decision processes. In: 2012 Ninth International Conference on Quantitative Evaluation of Systems, pp. 84–93. IEEE (2012)

[HT13] Hartmanns, A., Timmer, M.: On-the-fly confluence detection for statistical model checking. In: Brat, G., Rungta, N., Venet, A. (eds.) NFM 2013. LNCS, vol. 7871, pp. 337–351. Springer, Heidelberg (2013)

[JLS12] Jegourel, C., Legay, A., Sedwards, S.: Cross-Entropy Optimisation of Importance Sampling Parameters for Statistical Model Checking. In: Madhusudan, P., Seshia, S.A. (eds.) CAV 2012. LNCS, vol. 7358, pp. 327–342. Springer, Heidelberg (2012)

[JLS13] Jegourel, C., Legay, A., Sedwards, S.: Importance splitting for statistical model checking rare properties. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 576–591. Springer, Heidelberg (2013)

[JLS14] Jegourel, C., Legay, A., Sedwards, S.: An effective heuristic for adaptive importance splitting in statistical model checking. In: Margaria, T., Steffen, B. (eds.) ISoLA 2014, Part II. LNCS, vol. 8803, pp. 143–159. Springer, Heidelberg (2014)

[KM53] Kahn, H., Marshall, A.W.: Methods of Reducing Sample Size in Monte Carlo Computations. Operations Research 1(5), 263–278 (1953)

[LP12] Lassaigne, R., Peyronnet, S.: Approximate planning and verification for large Markov decision processes. In: Proceedings of the 27th Annual ACM Symposium on Applied Computing, pp. 1314–1319. ACM (2012)

[LS91] Larsen, K.G., Skou, A.: Bisimulation through probabilistic testing. Inf. Comput. 94(1), 1–28 (1991)

[LS14] Legay, A., Sedwards, S.: Lightweight Monte Carlo verification of Markov decision processes (submitted, 2014)

[Oka59] Okamoto, M.: Some inequalities relating to the partial sum of binomial probabilities. Annals of the Institute of Statistical Mathematics 10, 29–35 (1959)

[Put94] Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. Wiley-Interscience (1994)

[Rid10] Ridder, A.: Asymptotic optimality of the cross-entropy method for markov chain problems. Procedia Computer Science 1(1), 1571–1578 (2010)

[SVA04] Sen, K., Viswanathan, M., Agha, G.: Statistical model checking of black-box probabilistic systems. In: Alur, R., Peled, D.A. (eds.) CAV 2004. LNCS, vol. 3114, pp. 202–215. Springer, Heidelberg (2004)

[SVA05a] Sen, K., Viswanathan, M., Agha, G.A.: Vesta: A statistical model-checker and analyzer for probabilistic systems. In: QEST, pp. 251–252. IEEE Computer Society (2005)

[SVA05b] Sen, K., Viswanathan, M., Agha, G.: On statistical model checking of stochastic systems. In: Etessami, K., Rajamani, S.K. (eds.) CAV 2005. LNCS, vol. 3576, pp. 266–280. Springer, Heidelberg (2005)

[Wal45] Wald, A.: Sequential tests of statistical hypotheses. Annals of Mathematical Statistics 16(2), 117–186 (1945)

[Whi85] White, D.J.: Real applications of Markov decision processes. Interfaces 15(6), 73–83 (1985)

[Whi88] White, D.J.: Further real applications of Markov decision processes. Interfaces 18(5), 55–61 (1988)

[Whi93] White, D.J.: A survey of applications of Markov decision processes. Journal of the Operational Research Society 44(11), 1073–1096 (1993)

[WHL14] Wognsen, E.R., Hansen, R.R., Larsen, K.G.: Battery-aware scheduling of mixed criticality systems. In: Margaria, T., Steffen, B. (eds.) ISoLA 2014, Part II. LNCS, vol. 8803, pp. 208–222. Springer, Heidelberg (2014)

[You05a] Younes, H.L.S.: Verification and Planning for Stochastic Processes with Asynchronous Events. PhD thesis, Carnegie Mellon (2005)

[You05b] Younes, H.L.S.: Ymer: A statistical model checker. In: Etessami, K., Rajamani, S.K. (eds.) CAV 2005. LNCS, vol. 3576, pp. 429–433. Springer, Heidelberg (2005)

[Zul14] Zuliani, P.: Statistical model checking for biological applications. CoRR, abs/1405.2705 (2014)

An Effective Heuristic for Adaptive Importance Splitting in Statistical Model Checking

Cyrille Jegourel, Axel Legay, and Sean Sedwards

{cyrille.jegourel,axel.legay,sean.sedwards}@inria.fr

Abstract Statistical model checking avoids the intractable growth of states associated with numerical model checking by estimating the probability of a property from simulations. Rare properties pose a challenge because the relative error of the estimate is unbounded. In [13] we describe how importance splitting may be used with SMC to overcome this problem. The basic idea is to decompose a logical property into nested properties whose probabilities are easier to estimate. To improve performance it is desirable to decompose the property into many equiprobable *levels*, but logical decomposition alone may be too coarse.

In this article we make use of the notion of a *score function* to improve the granularity of a logical property. We show that such a score function may take advantage of heuristics, so long as it also rigorously respects certain properties. To demonstrate our importance splitting approach we present an optimal adaptive importance splitting algorithm and an heuristic score function. We give experimental results that demonstrate a significant improvement in performance over alternative approaches.

1 Introduction

Model checking offers the possibility to automatically verify the correctness of complex systems or detect bugs [7]. In many practical applications it is also useful to quantify the probability of a property (e.g., system failure), so the concept of model checking has been extended to probabilistic systems [1]. This form is frequently referred to as *numerical* model checking.

To give results with certainty, numerical model checking algorithms effectively perform an exhaustive traversal of the states of the system. In most real applications, however, the state space is intractable, scaling exponentially with the number of independent state variables (the 'state explosion problem' [6]). Abstraction and symmetry reduction may make certain classes of systems tractable, but these techniques are not generally applicable. This limitation has prompted the development of *statistical* model checking (SMC), which employs an executable model of the system to estimate the probability of a property from simulations.

SMC is a Monte Carlo method which takes advantage of robust statistical techniques to bound the error of the estimated result (e.g., [18,22]). To quantify a property it is necessary to observe the property, while increasing the number of observations generally increases the confidence of the estimate. Rare properties are often highly relevant to system performance (e.g., bugs and system failure are required to be rare) but pose a problem for statistical model checking because they are difficult to observe. Fortunately, rare event techniques such as *importance sampling* [14,16] and *importance splitting* [15,16,20] may be successfully applied to statistical model checking.

Importance sampling and importance splitting have been widely applied to specific simulation problems in science and engineering. Importance sampling works by estimating a result using weighted simulations and then compensating for the weights. Importance splitting works by reformulating the rare probability as a product of less rare probabilities conditioned on levels that must be achieved.

Recent work has explicitly considered the use of importance sampling in the context of statistical model checking [19,11,2,12]. Some limitations of importance sampling are discussed in [13]. In particular, it remains an open problem to quantify the performance of the importance sampling *change of measure*. A further numerical challenge arises from properties and systems that require very long simulations. In these cases the change of measure is only very subtly different to the original measure and may be difficult to represent with standard fixed length data types.

Earlier work [21,10] extended the original applications of importance splitting to more general problems of computational systems. In [13] we proposed the use of importance splitting for statistical model checking, specifically linking the concept of levels and score functions to temporal logic. In that work we considered two algorithms which make use of a fixed number of simulations per level. The first algorithm is based on fixed levels, chosen a priori, whose probabilities may not be equal. The second algorithm finds levels adaptively, evenly distributing the probability between them. In so doing, the adaptive algorithm reduces the relative error of the final estimate.

In what follows we show that importance splitting is an effective technique for statistical model checking and discuss the important role of the score function. We motivate and demonstrate the need for fine grained score functions that allow adaptive algorithms to find optimal levels. We then present a fine grained heuristic score function and an optimal adaptive importance splitting algorithm that improve on the performance of previous algorithms. We perform a set of experiments to illustrate both advantages and drawbacks of the technique.

The remainder of the paper is organised as follows. Section 2 defines the notation used in the sequel and introduces the basic notions of SMC applied to rare properties. Section 3 introduces the specific notions of importance splitting and score functions. Section 4 gives our importance splitting algorithms, while Section 5 illustrates their use on the dining philosophers protocol.

2 Statistical Model Checking Rare Events

We consider stochastic discrete-event systems. This class includes any stochastic process that can be thought of as occupying a single state for a duration of time before an instantaneous transition to a new state. In particular, we consider systems described by discrete and continuous time Markov chains. Sample execution paths can be generated by efficient discrete-event simulation algorithms (e.g., [9]). Execution paths are sequences of the form $\omega = s_0 \xrightarrow{t_0} s_1 \xrightarrow{t_1} s_2 \xrightarrow{t_2} ...$, where each s_i is a state of the model and $t_i \in \mathbb{R} > 0$ is the time spent in the state s_i (the delay time) before moving to the state s_{i+1}. In the case of discrete time, $t_i \equiv 1, \forall i$. When we are not interested by the times of jump epochs, we denote a path $\omega = s_0 s_1....$ The length of path ω includes the initial state and is denoted $|\omega|$. A prefix of ω is a sequence $\omega_{\leq k} = s_0 s_1 ... s_k$ with $k < |\omega| \in \mathbb{N}$. We denote by $\omega_{\geq k}$ the suffix of ω starting at s_k.

In the context of SMC we consider properties specified in bounded temporal logic, which may evaluate to true or false when applied to a specific path. Given a stochastic system and a bounded temporal logic property φ, our objective is to calculate the probability γ that an arbitrary execution trace ω satisfies φ, denoted $\gamma = \mathrm{P}(\omega \models \varphi)$. Let Ω be the set of paths induced by the initial state of the system, with $\omega \in \Omega$ and f a probability measure over Ω. To decide the truth of a particular trace ω', we define a model checking function z from Ω to $\{0, 1\}$ that takes the value 1 if $\omega' \models \varphi$ and 0 if $\omega' \not\models \varphi$. Thus,

$$\gamma = \int_{\Omega} z(\omega)\, \mathrm{d}f \qquad \text{and} \qquad \tilde{\gamma} = \frac{1}{N} \sum_{i=1}^{N} z(\omega_i)$$

N denotes the number of simulations and ω_i is sampled according to f. Note that the estimate $\tilde{\gamma}$ is distributed according to a binomial distribution with parameters N and γ. Hence $\mathrm{Var}(\tilde{\gamma}) = \gamma(1-\gamma)/N$ and for $\gamma \to 0$, $\mathrm{Var}(\tilde{\gamma}) \approx \gamma/N$.

When a property is not rare there are useful bounding formulae (e.g., the Chernoff bound [18]) that relate *absolute* error, confidence and the required number of simulations to achieve them. As the property becomes rarer, however, absolute error ceases to be useful and it is necessary to consider *relative* error, defined as the standard deviation of the estimate divided by its expectation. For the binomial random variable described above the relative error of an estimate is given by $\sqrt{\gamma(1-\gamma)}/N\gamma$, which is unbounded as $\gamma \to 0$. In standard Monte Carlo simulation, γ is the expected fraction of executions in which the rare event will occur. If the number of simulation runs is significantly less than $1/\gamma$, as is often necessary when γ is very small, no occurrences of the rare property will likely be observed. A number of simulations closer to $100/\gamma$ is desirable to obtain a reasonable estimate. Hence, importance sampling and importance splitting have been developed to reduce the number of simulations required or, equivalently, to reduce the variance of the rare event and so achieve greater confidence for a given number of simulations. In this work we focus on importance splitting.

3 Importance Splitting

The earliest application of importance splitting is perhaps that of [14,15], where it is used to calculate the probability that neutrons pass through certain shielding materials. This physical example provides a convenient analogy for the more general case. The system comprises a source of neutrons aimed at one side of a shield of thickness T. It is assumed that neutrons are absorbed by random interactions with the atoms of the shield, but with some small probability γ it is possible for a neutron to pass through the shield. The distance travelled in the shield can then be used to define a set of increasing levels $l_0 = 0 < l_1 < l_2 < \cdots < l_n = T$ that may be reached by the paths of neutrons. Importantly, reaching a given level implies having reached all the lower levels. Though the overall probability of passing through the shield is small, the probability of passing from one level to another can be made arbitrarily close to 1 by reducing the distance between the levels.

These concepts can be generalised to simulation models of arbitrary systems, where a path is a simulation trace. By denoting the abstract level of a path as l, the probability of reaching level l_i can be expressed as $\mathrm{P}(l > l_i) = \mathrm{P}(l > l_i \mid l > l_{i-1})\mathrm{P}(l > l_{i-1})$. Defining $\gamma = \mathrm{P}(l > l_n)$ and observing $\mathrm{P}(l > l_0) = 1$, it is possible to write

$$\gamma = \prod_{i=1}^{n} \mathrm{P}(l > l_i \mid l > l_{i-1}) \tag{1}$$

Each term of the product (1) is necessarily greater than or equal to γ. The technique of importance splitting thus uses (1) to decompose the simulation of a rare event into a series of simulations of conditional events that are less rare. There have been many different implementations of this idea, but a generalised procedure is as follows.

Assuming a set of increasing levels is defined as above, at each level a number of simulations are generated, starting from a distribution of initial states that correspond to reaching the current level. The procedure starts by estimating $\mathrm{P}(l \geq l_1 | l \geq l_0)$, where the distribution of initial states for l_0 is usually given (often a single state). Simulations are stopped as soon as they reach the next level; the final states becoming the empirical distribution of initial states for the next level. Simulations that do not reach the next level (or reach some other stopping criterion) are discarded. In general, $\mathrm{P}(l \geq l_i | l \geq l_{i-1})$ is estimated by the number of simulation traces that reach l_i, divided by the total number of traces started from l_{i-1}. Simulations that reached the next level are continued from where they stopped. To avoid a progressive reduction of the number of simulations, the generated distribution of initial states is sampled to provide additional initial states for new simulations, thus replacing those that were discarded.

In physical and chemical systems, distances and quantities may provide a natural notion of level that can be finely divided. In the context of model checking arbitrary systems, variables may be Boolean and temporal logic properties may not contain an obvious notion of level. To apply importance splitting to statistical model checking it is necessary to define a set of levels based on a sequence of logical properties, φ_i, that have the characteristic

$$\varphi = \varphi_n \Rightarrow \varphi_{n-1} \Rightarrow \cdots \Rightarrow \varphi_0 \tag{2}$$

Each φ_i is a strict restriction of the property φ_{i-1}, formed by the conjunction of φ_i with property ψ_i, such that $\varphi_i = \varphi_{i-1} \wedge \psi_i$, with $\varphi_0 \equiv \top$. Hence, φ_i can be written $\varphi_i = \bigwedge_{j=1}^{i} \psi_j$. This induces a strictly nested sequence of sets of paths $\Omega_i \subseteq \Omega$:

$$\Omega_n \subset \Omega_{n-1} \subset \cdots \subset \Omega_0$$

where $\Omega_i = \{\omega \in \Omega : \omega \models \varphi_i\}$, $\Omega_0 \equiv \Omega$ and $\forall \omega \in \Omega, \omega \models \varphi_0$. Thus, for arbitrary $\omega \in \Omega$,

$$\gamma = \prod_{i=1}^{n} \mathrm{P}(\omega \models \varphi_i \mid \omega \models \varphi_{i-1}),$$

which is analogous to (1).

A statistical model checker works by constructing an automaton to accept traces that satisfy the specified property. In the context of SMC, importance splitting requires that the state of this automaton be included in the final state of a trace that reaches a given level. In practice, this means storing the values of the counters of the loops that implement the time bounded temporal operators.

The choice of levels is crucial to the effectiveness of importance splitting. To minimise the relative variance of the final estimate it is desirable to choose levels that make $\mathrm{P}(\omega \models \varphi_i \mid \omega \models \varphi_{i-1})$ the same for all i (see, e.g., [8]). A simple decomposition of a property may give levels with widely divergent conditional probabilities, hence [13] introduces the concept of a *score function* and techniques that may be used to increase the possible granularity of levels. Given sufficient granularity, a further challenge is to define the levels. In practice these might be guessed or found by trial and error, but Section 4 gives algorithms that find optimal levels adaptively.

Score Functions

The goal of a score function S is to discriminate good paths from bad with respect to the property of interest. This is often expressed as a function from paths to $\mathbb{R}$, assigning higher values to paths which more nearly satisfy the overall property. Standard statistical model checking can be seen as a degenerate case of splitting, in the sense that computing $P(\omega \models \varphi)$ is equivalent to compute $P(S(\omega) \geq 1)$ with the functional equality $S = z$, where z is the Bernoulli distributed model checking function.

Various ways to decompose a temporal logic property are proposed in [13]. Given a sequence of nested properties $\varphi_0 \Leftarrow \varphi_1 \Leftarrow \cdots \Leftarrow \varphi_n = \varphi$, one may design a function which directly correlates logic to score. For example, a simple score function may be defined as follows:

$$S(\omega) = \sum_{k=1}^{n} \mathbb{1}(\omega \models \varphi_k)$$

$\mathbb{1}(\cdot)$ is an indicator function taking the value 1 when its argument is true and 0 otherwise.

Paths that have a higher score are clearly better because they satisfy more of the overall property. However in many applications the property of interest may not have a suitable notion of levels to exploit; the logical levels may be too coarse or may distribute the probability unevenly. For example, given the dining philosophers problem presented in section 5, we know that from a thinking state, a philosopher must pick one fork and then a second one before eating, but there is no obvious way of creating a finer score function from these logical subproperties and actually, the probability of satisfying a subproperty from a state such that the previous subproperty is satisfied is too low (about 0.06, see Table 1). For these cases it is necessary to design a more general score function which maps a larger sequence of nested set of paths to a set of nested intervals of $\mathbb{R}$.

Denoting an arbitrary path by ω and two path prefixes by ω' and ω'', an ideal score function S satisfies the following property:

$$S(\omega') \geq S(\omega'') \iff \mathrm{P}(\omega \models \varphi \mid \omega') \geq \mathrm{P}(\omega \models \varphi \mid \omega'') \tag{3}$$

Intuitively, (3) states that prefix ω' has greater score than prefix ω'' if and only if the probability of satisfying φ with paths having prefix ω' is greater than the probability of satisfying φ with paths having prefix ω''.

Designing a score function which satisfies (3) is generally infeasible because it requires a huge analytical work based on a detailed knowledge of the system and, in particular, of the probability of interest. However, the minimum requirement of a score function is much less demanding. Given a set of nested properties $\varphi_1, \dots \varphi_i, \dots, \varphi_n$ satisfying (2), the requirement of a score function is that $\omega \models \varphi_i \iff S(\omega) \geq \tau_i$, with $\tau_i > \tau_{i-1}$ a monotonically increasing set of numbers called thresholds. Even a simple score function with few levels (e.g., $n = 2$) could nevertheless provide an unbiased estimate with a likely smaller number of traces than a standard Monte Carlo estimation.

When no formal levels are available, an effective score function may still be defined using heuristics that only loosely correlate increasing score with increasing probability of satisfying the property. In particular, a score function based on coarse logical levels may be given increased granularity by using heuristics between the levels. For example, a time bounded property, not explicitly correlated to time, may become increasingly less likely to be satisfied as time runs out (i.e., with increasing path length). A plausible heuristic in this case is to assign higher scores to shorter paths. A similar heuristic has been used for importance sampling, under the assumption that the mass of probability in the optimal change of measure is centred on short, direct paths [19]. In the context of importance splitting, the assumption is that shorter paths that satisfy the sub-property at one level are more likely to satisfy the sub-property at the next level because they have more time to do so. We make use of this heuristic in Section 4.

4 Importance Splitting Algorithms

We give three importance splitting pseudo-algorithms based on [4]; in the first one, levels are fixed and defined a priori, the number of levels is an input of the algorithm; in the second one, levels are found adaptively with respect to a predefined probability, the number of levels is a random variable and is not an input anymore; the third one is an extension of the second where the probability to cross a level from a previous stage is set to its maximum. By N we denote the number of simulations performed at each level. Thresholds, denoted τ, are usually but not necessarily defined as values of score function $S(\omega)$, where ω is a path. τ_φ is the minimal threshold such that $S(\omega) \geq \tau_\varphi \iff \omega \models \varphi$. τ_k is the k^{th} threshold and ω_i^k is the i^{th} simulation on level k. $\tilde{\gamma}_k$ is the estimate of γ_k, the k^{th} conditional probability $\mathrm{P}(S(\omega) \geq \tau_k \mid S(\omega) \geq \tau_{k-1})$.

4.1 Fixed Level Algorithm

The fixed level algorithm follows from the general description given in Section 3. Its advantages are that it is simple, it has low computational overhead and the resulting estimate is unbiased. Its disadvantage is that the levels must often be guessed by trial and error – adding to the overall computational cost.

In Algorithm 1, $\tilde{\gamma}$ is an unbiased estimate (see, e.g., [8]). Furthermore, from Proposition 3 in [4], we can deduce the following $(1-\alpha)$ confidence interval:

$$CI = \left[\tilde{\gamma}\left(\frac{1}{1+\frac{z_\alpha \sigma}{\sqrt{N}}}\right), \tilde{\gamma}\left(\frac{1}{1-\frac{z_\alpha \sigma}{\sqrt{N}}}\right)\right] \qquad \text{with} \qquad \sigma^2 \geq \sum_{k=1}^{M} \frac{1-\gamma_k}{\gamma_k}, \tag{4}$$

where z_α is the $1-\frac{\alpha}{2}$ quantile of the standard normal distribution. Hence, with confidence $100(1-\alpha)\%$, $\gamma \in CI$. For any fixed M, the minimisation problem

$$\min \sum_{k=1}^{M} \frac{1-\gamma_k}{\gamma_k} \qquad \text{with constraint} \qquad \prod_{k=1}^{M} \gamma_k = \gamma$$

implies that σ is reduced by making all γ_k equal.

For given γ, this motivates fine grained score functions. When it is not possible to define γ_k arbitrarily, the confidence interval may nevertheless be reduced by increasing N. The inequality for σ arises because the independence of initial states diminishes with increasing levels: unsuccessful traces are discarded and new initial states are drawn from successful traces. Several possible ways to minimise these dependence effects are proposed in [4]. In the following, for the sake of simplicity, we assume that this goal is achieved. In the confidence interval, σ is estimated by the square root of $\sum_{k=1}^{M} \frac{1-\tilde{\gamma}_k}{\tilde{\gamma}_k}$.

4.2 Adaptive Level Algorithm

The cost of finding good levels must be included in the overall computational cost of importance splitting. An alternative to trial and error is to use an adaptive level algorithm that discovers its own optimal levels.

Algorithm 2 is an adaptive level importance splitting algorithm presented first in [5]. It works by pre-defining a fixed number N_k of simulation traces to retain at each level. With the exception of the last level, the conditional probability of each

Algorithm 1: Fixed levels

```
Let $(\tau_k)_{1 \leq k \leq M}$ be the sequence of thresholds with $\tau_M = \tau_\varphi$
Let *stop* be a termination condition
$\forall j \in \{1, \dots, N\}$, set prefix $\tilde{\omega}_j^1 = \epsilon$ (empty path)
**for** $1 \leq k \leq M$ **do**
    $\forall j \in \{1, \dots, N\}$, using prefix $\tilde{\omega}_j^k$, generate path $\omega_j^k$ until $(S(\omega_j^k) \geq \tau_k) \vee stop$
    $I_k = \{\forall j \in \{1, \dots, N\} : S(\omega_j^k) \geq \tau_k\}$
    $\tilde{\gamma}_k = \frac{|I_k|}{N}$
    $\forall j \in I_k$, $\tilde{\omega}_j^{k+1} = \omega_j^k$
    $\forall j \notin I_k$, let $\tilde{\omega}_j^{k+1}$ be a copy of $\omega_i^k$ with $i \in I_k$ chosen uniformly randomly
$\tilde{\gamma} = \prod_{k=1}^{M} \tilde{\gamma}_k$
```

Algorithm 2: Adaptive levels

```
Let $\tau_\varphi = \min\{S(\omega) \mid \omega \models \varphi\}$ be the minimum score of paths that satisfy $\varphi$
Let $N_k$ be the pre-defined number of paths to keep per iteration
$k = 1$
$\forall j \in \{1, \dots, N\}$, generate path $\omega_j^k$
**repeat**
    Let $T = \left\{S(\omega_j^k), \forall j \in \{1, \dots, N\}\right\}$
    Find maximum $\tau_k \in T$ such that $|\{\tau \in T : \tau > \tau_k\}| \geq N - N_k$
    $\tau_k = \min(\tau_k, \tau_\varphi)$
    $I_k = \{j \in \{1, \dots, N\} : S(\omega_j^k) > \tau_k\}$
    $\tilde{\gamma}_k = \frac{|I_k|}{N}$
    $\forall j \in I_k$, $\omega_j^{k+1} = \omega_j^k$
    **for** $j \notin I_k$ **do**
        choose uniformly randomly $l \in I_k$
        $\tilde{\omega}_j^{k+1} = \max_{|\omega|} \left\{\omega \in pref(\omega_l^k) : S(\omega) < \tau_k\right\}$
        generate path $\omega_j^{k+1}$ with prefix $\tilde{\omega}_j^{k+1}$
    $M = k$
    $k = k + 1$
**until** $\tau_k > \tau_\varphi$;
$\tilde{\gamma} = \prod_{k=1}^{M} \tilde{\gamma}_k$
```

level is then nominally N_k/N. Making N_k all equal minimizes the overall relative variance and is only possible if the score function has sufficient granularity.

Use of the adaptive algorithm may lead to gains in efficiency (no trial and error, reduced overall variance), however the final estimate has a bias of order $\frac{1}{N}$, i.e., $\mathrm{E}(\tilde{\gamma}) = \gamma\left(1 + \mathcal{O}(N^{-1})\right)$. The overestimation (potentially not a problem when estimating rare critical failures) is negligible with respect to σ, such that the confidence interval remains that of the fixed level algorithm. Furthermore, under some regularity conditions, the bias can be asymptotically corrected. The estimate of γ has the form $r_0{\gamma_0}^{M_0}$, with $M_0 = M-1$, $r_0 = \gamma{\gamma_0}^{-M_0}$ and $\frac{E[\tilde{\gamma}]-\gamma}{\gamma} \sim \frac{M_0}{N}\frac{1-\gamma_0}{\gamma_0}$ when N goes to infinity. Using the expansion

$$\tilde{\gamma} = \gamma\left(1 + \frac{1}{\sqrt{N}}\sqrt{M_0\frac{1-\gamma_0}{\gamma_0} + \frac{1-r_0}{r_0}}Z + \frac{1}{N}M_0\frac{1-\gamma_0}{\gamma_0} + o\left(\frac{1}{N}\right)\right),$$

with Z a standard normal variable, $\tilde{\gamma}$ is corrected by dividing it by $1 + \frac{M_0(1-\gamma_0)}{N\gamma_0}$. See [4] for more details.

4.3 Optimized Adaptive Level Algorithm

Algorithm 3 defines an optimized adaptive level importance splitting algorithm. The variance of the estimate $\tilde{\gamma}$ is:

$$Var(\tilde{\gamma}) = \frac{p^2}{N}\left(n_0\frac{1-\gamma_0}{\gamma_0} + \frac{1-r_0}{r_0} + o(N^{-1})\right)$$

and the function $f : \gamma_0 \longmapsto \frac{1-\gamma_0}{-\gamma_0\, log\, \gamma_0}$ is strictly decreasing on $]0,1[$. Increasing γ_0 therefore decreases the variance. Ideally, this value is $\gamma_0 = 1 - \frac{1}{N}$ but it is more realistic to fix this value for each iteration k at $\gamma_0 = 1 - \frac{N_k}{N}$, with N_k the number of paths achieving the minimal score. Another advantage of this optimized version is that, although the number of steps before the algorithm terminates is more important, we only rebranch a few discarded traces (ideally only 1) per iteration.

Remark about Rebranching. At the end of iteration k, we end up with an estimate of γ_k and an approximation $\tilde{l}_k$ of the first entrance state distribution into level k. The discarded traces must be rebranched over a successful prefix with respect to distribution $\tilde{l}_k$. In practise, to decrease the variance, we do not pick uniformly an index of a successful path but a cycle of indexes of successful paths. In doing so we avoid the unlikely but possible rebranching of all the discarded traces from the same state.

Let I_k and J_k be respectively the sets of indexes of successful and discarded prefixes. We denote by respectively $I_k(j)$ and $J_k(j)$ the j-th index of I_k and J_k. Let $\mathfrak{S}_{|I_k|}$ be the set of permutations of $\{1, \cdots, |I_k|\}$ and ι an element of $\mathfrak{S}_{|I_k|}$.

We then build randomly a $|J_k|$-length vector $\tilde{J}_k$ with elements of I_k. We choose uniformly cycle ι of $\mathfrak{S}_{|I_k|}$ and repeat the chosen cycle if $N - |I_k| \geq |I_k|$. The first

Algorithm 3: Optimized adaptive levels

```
Let $\tau_\varphi = \min\{S(\omega) \mid \omega \models \varphi\}$ be the minimum score of paths that satisfy $\varphi$
$k = 1$
$\forall j \in \{1, \ldots, N\}$, generate path $\omega_j^k$
repeat
    Let $T = \left\{S(\omega_j^k), \forall j \in \{1, \ldots, N\}\right\}$
    $\tau_k = \min T$
    $\tau_k = \min(\tau_k, \tau_\varphi)$
    $I_k = \{j \in \{1, \ldots, N\} : S(\omega_j^k) > \tau_k\}$
    $\tilde{\gamma}_k = \frac{|I_k|}{N}$
    $\forall j \in I_k$, $\omega_j^{k+1} = \omega_j^k$
    for $j \notin I_k$ do
        choose uniformly randomly $l \in I_k$
        $\tilde{\omega}_j^{k+1} = \max_{|\omega|} \left\{\omega \in \mathit{pref}(\omega_l^k) : S(\omega) < \tau_k\right\}$
        generate path $\omega_j^{k+1}$ with prefix $\tilde{\omega}_j^{k+1}$
    $M = k$
    $k = k + 1$
until $\tau_k > \tau_\varphi$;
$\tilde{\gamma} = \prod_{k=1}^{M} \tilde{\gamma}_k$
```

$|J_k|$ elements are the respective elements of $\tilde{J}_k$. Finally, we assign to discarded prefix $\omega_{J_k(j)}$ the successful prefix $\omega_{\tilde{J}_k(j)} = \omega_{I_k((\iota(j)-1 \text{ modulo } |I_k|)+1)}$.

This circular sampling has the advantage to resample perfectly with respect to distribution $\tilde{l}_k$.

5 Case Study: Dining Philosophers Protocol

We have adapted a case study from the literature to illustrate the use of heuristic-based score functions and of the optimized adaptive splitting algorithm with statistical model checking. We have defined a rare event in the well known probabilistic solution [17] of Dijkstra's dining philosophers problem . In this example, there are no natural counters to exploit, so levels must be constructed by considering 'lumped' states.

A number of philosophers sit at a circular table with an equal number of chopsticks; a chopstick being placed within reach of two adjacent philosophers. Philosophers think and occasionally wish to eat from a communal bowl. To eat, a philosopher must independently pick up two chopsticks: one from the left and one from the right. Having eaten, the philosopher replaces the chopsticks and returns to thinking. A problem of concurrency arises because a philosopher's neighbour(s) may have already taken the chopstick(s). Lehmann and Rabin's solution [17] is to allow the philosophers to make probabilistic choices.

We consider a model of 150 'free' philosophers [17]. The number of states in the model is more than 10^{177}; 10^{97} times more than the estimated number of protons in the universe. The possible states of an individual philosopher can be abstracted to those shown in Fig. 1.

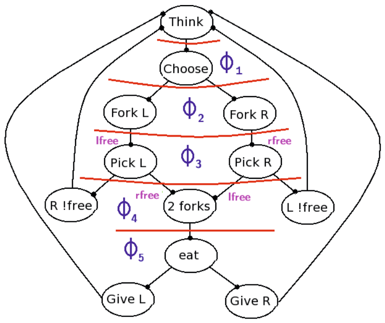


Fig. 1. Abstract dining philosopher

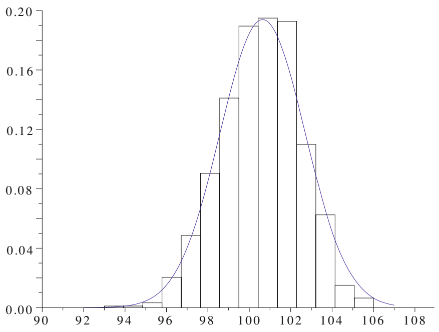


Fig. 2. Empirical number of levels

Think is the initial state of all philosophers. In state Choose, the philosopher makes a choice of fork he will try to get first. The transitions labelled by lfree or rfree in Fig. 1 are dependent on the availability of respectively left or right chopsticks. All transitions are controlled by stochastic rates and made in competition with the transitions of other philosophers. With increasing numbers of philosophers, it is increasingly unlikely that a specific philosopher will be satisfied (i.e., that the philosopher will reach the state eat) within a given number of steps from the initial state. We thus define a rare property $\varphi = \mathbf{F}^t\mathsf{eat}$, with t initially 30, denoting the property that a given philosopher will reach state eat within 30 steps. Thus, using the states of the abstract model, we decompose φ into nested properties $\varphi_0 = \mathbf{F}^t\mathsf{Think} = \top$, $\varphi_1 = \mathbf{F}^t\mathsf{Choose}$, $\varphi_2 = \mathbf{F}^t\mathsf{Try}$, $\varphi_3 = \mathbf{F}^t\mathsf{1^{st}stick}$, $\varphi_4 = \mathbf{F}^t\mathsf{2^{nd}stick}$, $\varphi_5 = \mathbf{F}^t\mathsf{eat}$. The red lines crossing the transitions indicate these formal levels on the graph.

Monte Carlo Simulations with PLASMA Statistical Model Checker. With such a large state space it is not possible to obtain a reference result with numerical model checking. We therefore performed extensive Monte Carlo simulations using the parallel computing capability of the PLASMA statistical model checker [11,3]. The experiment generated 300 million samples using 255 cores and took about 50 minutes. Our reference probability is thus approximately equal to 1.59×10^{-6} with 95%-confidence interval $\left[1.44 \times 10^{-6}; 1.72 \times 10^{-6}\right]$.

Recall and Experiment Protocol. Table 1 recalls, given a score function, that the parameters of each algorithm for an experiment are the number n of simulations used at the first iteration and the distance between levels (usually constant) in the fixed level algorithm or a probability between levels for the adaptive algorithms.

Table 1. Parameters in each ISp algorithm

initial parameters n, $\tau_k - \tau_{k-1}$, γ_0	fixed alg.	adaptive alg.	optimized alg
Number n of path at first iteration	YES	YES	YES
Step between levels ($\tau_k - \tau_{k-1}$)	YES	NO	NO
conditionnal probability γ_0	NO	YES	NO

Note that the conditionnal probability in the optimized algorithm is a function of n more than an independent parameter.

Four types of importance splitting experiments are driven. The first one uses the simple score function and the fixed algorithm, the second uses the heuristic score function and the fixed-level algorithm (with different step values). The third algorithm uses the adaptive-level algorithm with different γ_0 parameters and finally the fourth set of experiments uses the optimized version of the adaptive algorithm.

For each set of experiments and chosen parameters, experiments are repeated 100 times in order to check the reliability of our results. In what follows, we remind which statistical notions are exploited and why:

– Number of experiments: used to estimate the variance of the estimator.
– Number of path per iteration: it is a parameter of the algorithm, equal to the number of paths that we use to estimate a conditionnal probability.
– Number of levels: known in the fixed algorithm, variable in the adaptive algorithms. In the second case, an average is provided.
– Time in seconds: the average of the 100 experiments is provided.
– The mean estimate is the estimator $\tilde{\gamma}$ of the probability of interest. The average of the 100 estimators is provided.
– The relative standard deviation of $\tilde{\gamma}$ is estimated with the 100 final estimators γ. A reliable estimator must have a low relative standard deviation (roughly ≤ 0.3).
– The mean value of γ_k is the average of the mean values of the conditionnal probabilities in an experiment. It is variable in the fixed algorithm and supposed to be a constant γ_0 in the adaptive algorithms. Because of the discreteness of the score function, the value is only almost constant and slightly lower than γ_0.
– The relative standard deviation of γ_k is the average of the relative standard deviations of the conditionnal probabilities in an experiment. By construction, the value in the adaptive algorithms must be low.

Comparison between Logical and Heuristic Score Function. Let ω be a path of length $t = 30$. For each prefix $\omega_{\leq j}$ of length j, we define the following function:

$$\Psi(\omega_{\leq j}) = \sum_{k=0}^{n} \mathbb{1}(\omega_{\leq j} \models \varphi_k) - \frac{\{\sum_{k=1}^{n} \mathbb{1}(\omega_{\leq j} \models \varphi_k)\} - j}{\sum_{k=1}^{n} \mathbb{1}(\omega_{\leq j} \models \varphi_k) - (t+1)}$$

We define score of ω as follows:

$$S(\omega) = \max_{1 \le j \le K} \Psi(\omega_{\le j})$$

In the following experiment this score function is defined for any path of length $t+1$, starting in the initial state 'all philosophers think'. The second term of Ψ is a number between 0 and 1, linear in j such that the function gives a greater score to paths which satisfy a greater number of sub-properties φ_k and discriminates between two paths satisfying the same number of sub-properties by giving a greater score to the shortest path. A score in $]i-1, i]$ implies that a prefix of the path satisfied at most φ_i. We then compare results with the simple score function $S(\omega) = \sum_{k=1}^{n} \mathbb{1}(\omega \models \varphi_k)$.

Table 2. Comparison between fixed-level algorithms

Statistics	Simple score function	Heuristic score function		
number of experiments	100	100	100	100
number of path per iteration	1000	1000	1000	1000
number of levels	5	20	40	80
Time in seconds (average)	6.95	13.42	16.64	21.56
mean estimate $\times 10^6$ (average)	0.01	0.59	1	1.37
mean value of $\tilde{\gamma_k}$	0.06	0.53	0.73	0.86
relative standard deviation of $\tilde{\gamma_k}$	1.04	0.36	0.22	0.15

The experiments are repeated 100 times in order to demonstrate and improve the reliability of the results. Each conditional probability γ_k is estimated with a sample of 1000 paths.

For simplicity we consider a linear growing of score thresholds when we use the fixed-level algorithm. The simple score function thresholds increase by 1 between each level. When using the heuristic score function, we performed three sets of experiments involving an increase of 0.2, 0.1 and 0.05 of the thresholds. These partitions imply respectively 5, 20, 40 and 80 levels.

Table 2 shows that the simple score function likely gives a strong underestimation. It is due to the huge decrease of value of conditional probabilities between the logical levels. All the estimated conditional probabilities are small and imply a large theoretical relative variance $(V(\tilde{\gamma})/E[\tilde{\gamma}])$. The final levels are difficult to cross and have probabilities close to 0. A sample size of 1000 paths is obviously not enough for the last step. On average $\tilde{\gamma}_5 = 0.003$ and in one case the last step is not satisfied by any trace, such that the estimate is equal to zero.

If a threshold is not often exceeded, it implies that traces will be rebranched from a very small set of first entrance states at the next level. This leads to significant relative variance between experiments. A further problem is that the conditional estimate is less efficient if γ_k is small. Increasing the number of evenly spaced levels decrease *a priori* more smoothly the conditional probabilities and reinforce the reliability of the results as soon as the relative standard deviation of conditional probabilities decreases enough. In the experiments, as expected, the mean value of conditional probabilities is positively correlated to the number of levels (respectively 0.06, 0.53, 0.73 and 0.86) and negatively correlated to the relative standard deviation of conditional probabilities. The results with 40 and 80 levels give results that are apparently close to the reference estimate, but are nevertheless consistently underestimates. This suggests that the number of simulations per level is too low.

Two questions arise: how to detect that the simulation is not efficient or robust and how to improve the results. In answer to the first, there are no general criteria for judging the quality of an importance splitting estimator. However, assuming that experiments are repeated a few times, a large relative error of the estimators (roughly ≥ 0.5), a very low value of conditional probability estimates, or a large relative error of conditional probability estimates (roughly ≥ 0.2) are good warnings. As for the second question, a way to improve results with the fixed level algorithm is simply to increase the number of paths per level or to increase the number of levels, for the reasons given above.

5.1 Comparison between Fixed and Adaptive Algorithm

The following section illustrates that adaptive algorithms give significantly more reliable results for slightly increased time. In the following set of experiments we use the adaptive algorithm with three predefined γ_0: 0.6, 0.75 and 0.9. Because of the granularity of the score function, conditional probabilities are not equal at each iteration, but their values are kept under control because their relative standard deviation does not vanish (≤ 0.2). We use 1000 sample paths per level and repeat the experiments 100 times.

Table 3. Comparison between adaptive algorithms

γ_0	0.6	0.75	0.9
number of experiments	100	100	100
number of path per iteration	1000	1000	1000
number of levels (average)	22	34	65
Time in seconds (average)	14.53	16.78	20.05
mean estimate $\times 10^6$ (average)	0.78	1.14	1.58
relative standard deviation of $\tilde{\gamma}$	0.26	0.25	0.23
mean value of $\tilde{\gamma_k}$	0.55	0.68	0.83
relative standard deviation of $\tilde{\gamma_k}$	0.2	0.16	0.12

As we increased the desired γ_0, the number of levels and time increase. However, the final estimate with $\gamma_0 = 0.9$ matches the Monte Carlo estimator and the relative standard deviation is minimized. In this experiment the number of levels found adaptively is on average 65. Even with mean value of conditional probabilities smaller than in the 80-fixed-level experiment, the results show better convergence, a slightly better speed and lower standard deviation.

5.2 Comparison with the Optimized Adaptive Algorithm

This section illustrates a set of experiments using the optimized adaptive algorithm. As previously, we repeated experiments 100 times to check reliability of our results. For each experiment we use a different number of initial paths: 100, 200, 500 and 1000. In order to give an idea of the gain of time, we also executed a Monte Carlo experiment using 10^7 paths. The 95%-confidence intervals are given by (4) for the importance splitting experiments and by the standard confidence interval $\left[\tilde{\gamma} \pm 1.96 \times \sqrt{\frac{\tilde{\gamma}(1-\tilde{\gamma})}{N}}\right]$ for Monte Carlo experiment. As the experiments are repeated several times, we approximate the relative standard deviation σ by the standard deviation of the estimates divided by the average of the estimates, instead of assuming full independence between levels and so taking $\sigma \approx \sum_{k=1}^{m} \frac{1-\gamma_k}{\gamma_k}$. Our approach is more pessimistic and in practise requires the experiment to be repeated a few times. However, even doing so, the results are much more accurate than the Monte Carlo approach. For example, 100 initial paths are used in the first experiment. Roughly speaking, the paths cross on average 100 other levels and only 11% are rebranched each time. So, only 1200 paths are generated and provide in less than 2 seconds an estimate and a confidence interval strictly included in the Monte Carlo confidence interval. This represents a gain greater than 10^4 with respect to the Monte Carlo experiment.

Table 4. Comparison between optimized adaptive algorithms

Statistics	Importance splitting				MC
number of experiments	100	100	100	100	1
number of path per iteration	100	200	500	1000	10 million
Time in seconds (average)	1.73	4.08	11.64	23.77	> 5 hours
mean estimate $\times 10^6$ (average)	1.52	1.59	1.58	1.65	1.5
standard deviation $\times 10^6$	1.02	0.87	0.5	0.38	0.39
95%-confidence interval $\times 10^6$	[1.34; 1.74]	[1.48; 1.72]	[1.54; 1.63]	[1.64; 1.66]	[0.74; 2.26]

Figure 2 illustrates empirically the convergence of the number of levels to a Gaussian with low variance (4.23) with respect to the mean of levels (100.65). Although this fact is only empirical, knowing that the variance is low has some importance whenever the time budget is critical for more extensive experiments.

6 Conclusion

We have presented an effective heuristic to improve the granularity of score functions for importance splitting. The logical properties used in statistical model checking can thus be decomposed into a greater number of levels, allowing the use of high performance adaptive algorithms. We have presented an optimized adaptive algorithm and shown how, in combination with our heuristic score function, it significantly improves on the performance of the alternatives. As future work, we would like to develop a Chernoff bound and sequential hypothesis test to complement the confidence interval presented here.

References

1. Baier, C., Katoen, J.-P.: Principles of Model Checking (Representation and Mind Series). The MIT Press (2008)
2. Barbot, B., Haddad, S., Picaronny, C.: Coupling and importance sampling for statistical model checking. In: Flanagan, C., König, B. (eds.) TACAS 2012. LNCS, vol. 7214, pp. 331–346. Springer, Heidelberg (2012)
3. Boyer, B., Corre, K., Legay, A., Sedwards, S.: PLASMA-lab: A flexible, distributable statistical model checking library. In: Joshi, K., Siegle, M., Stoelinga, M., D'Argenio, P. (eds.) QEST 2013. LNCS, vol. 8054, pp. 160–164. Springer, Heidelberg (2013)
4. Cérou, F., Del Moral, P., Furon, T., Guyader, A.: Sequential Monte Carlo for rare event estimation. Statistics and Computing 22, 795–808 (2012)
5. Cérou, F., Guyader, A.: Adaptive multilevel splitting for rare event analysis. Stochastic Analysis and Applications 25, 417–443 (2007)
6. Clarke, E., Emerson, E.A., Sifakis, J.: Model checking: algorithmic verification and debugging. Commun. ACM 52(11), 74–84 (2009)
7. Clarke Jr., E.M., Grumberg, O., Peled, D.A.: Model checking. MIT Press, Cambridge (1999)
8. Del Moral, P.: Feynman-Kac Formulae: Genealogical and Interacting Particle Systems with Applications. Probability and Its Applications. Springer (2004)
9. Gillespie, D.T.: Exact stochastic simulation of coupled chemical reactions. Journal of Physical Chemistry 81, 2340–2361 (1977)
10. Glasserman, P., Heidelberger, P., Shahabuddin, P., Zajic, T.: Multilevel splitting for estimating rare event probabilities. Oper. Res. 47(4), 585–600 (1999)
11. Jegourel, C., Legay, A., Sedwards, S.: A Platform for High Performance Statistical Model Checking – PLASMA. In: Flanagan, C., König, B. (eds.) TACAS 2012. LNCS, vol. 7214, pp. 498–503. Springer, Heidelberg (2012)
12. Jegourel, C., Legay, A., Sedwards, S.: Cross-Entropy Optimisation of Importance Sampling Parameters for Statistical Model Checking. In: Madhusudan, P., Seshia, S.A. (eds.) CAV 2012. LNCS, vol. 7358, pp. 327–342. Springer, Heidelberg (2012)
13. Jegourel, C., Legay, A., Sedwards, S.: Importance splitting for statistical model checking rare properties. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 576–591. Springer, Heidelberg (2013)
14. Kahn, H.: Random sampling (Monte Carlo) techniques in neutron attenuation problems. Nucleonics 6(5), 27 (1950)
15. Kahn, H., Harris, T.E.: Estimation of Particle Transmission by Random Sampling. In: Applied Mathematics. Series 12, vol. 5. National Bureau of Standards (1951)
16. Kahn, H., Marshall, A.W.: Methods of Reducing Sample Size in Monte Carlo Computations. Operations Research 1(5), 263–278 (1953)
17. Lehmann, D., Rabin, M.O.: On the Advantage of Free Choice: A Symmetric and Fully Distributed Solution to the Dining Philosophers Problem (Extended Abstract). In: Proc. 8th Ann. Symposium on Principles of Programming Languages, pp. 133–138 (1981)
18. Okamoto, M.: Some inequalities relating to the partial sum of binomial probabilities. Annals of the Institute of Statistical Mathematics 10, 29–35 (1959)

19. Reijsbergen, D., de Boer, P.-T., Scheinhardt, W., Haverkort, B.: Rare event simulation for highly dependable systems with fast repairs. Performance Evaluation 69(7-8), 336–355 (2012)
20. Rosenbluth, M.N., Rosenbluth, A.W.: Monte Carlo Calculation of the Average Extension of Molecular Chains. Journal of Chemical Physics 23(2) (February 1955)
21. Villén-Altamirano, M., Villén-Altamirano, J.: RESTART: A Method for Accelerating Rare Event Simulations. In: Cohen, J.W., Pack, C.D. (eds.) Queueing, Performance and Control in ATM, pp. 71–76. Elsevier (1991)
22. Wald, A.: Sequential Tests of Statistical Hypotheses. The Annals of Mathematical Statistics 16(2), 117–186 (1945)

A Formalism for Stochastic Adaptive Systems

Benoît Boyer, Axel Legay, and Louis-Marie Traonouez

Inria / IRISA, Campus de Beaulieu, 35042 Rennes CEDEX, France

Abstract. Complex systems such as systems of systems result from the combination of several components that are organized in a hierarchical manner. One of the main characteristics of those systems is their ability to adapt to new situations by modifying their architecture. Those systems have recently been the subject of a series of works in the software engineering community. Most of those works do not consider quantitative features. The objective of this paper is to propose a modeling language for adaptive systems whose behaviors depend on stochastic features. Our language relies on an extension of stochastic transition systems equipped with (1) an adaptive operator that allows to reason about the probability that a system has to adapt its architecture over time, and (2) dynamic interactions between processes. As a second contribution, we propose a contract-based extension of probabilistic linear temporal logic suited to reason about assumptions and guarantees of such systems. Our work has been implemented in the Plasma-Lab tool developed at Inria. This tool allows us to define stochastic adaptive systems with an extension of the Prism language, and properties with patterns. In addition, Plasma-Lab offers a simulation-based model checking procedure to reason about finite executions of the system. First experiments on a large case study coming from an industrial driven European project give encouraging results.

1 Context

Critical systems increasingly rely on dynamically adaptive programs to respond to changes in their physical environments. Reasoning about such systems require to design new verification techniques and formalisms that take this model of reactivity into account [7].

This paper proposes a complete formalism for the rigorous design of stochastic adaptive systems (SAS), whose components' behaviors and environment changes are represented via stochastic information. Adding some stochastic feature to components' models is more realistic, especially regarding the environment aspect, e.g. the probability of hardware failure, the fire frequency in a forest or a growing city population...

We view the evolution of our system as a sequence of views, each of them representing a topology of the system at a given moment of time. In our setting, views are represented by a combination of Markov chains, and stochastic adaptive transitions that describe the environment evolution as transitions between different views of the SAS (e.g. adding or removing components). Each view thus associates the new environment behaviour and a new system configuration

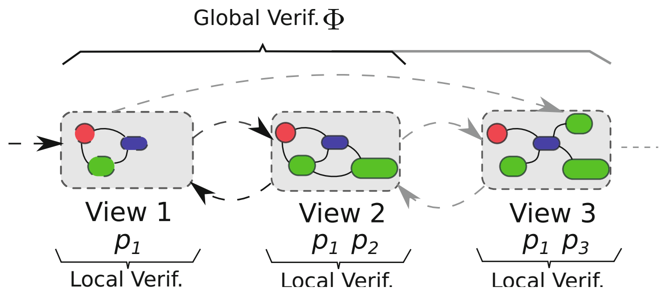


Fig. 1. Illustration of SAS Methodology

(a new topology, addition or suppression of system components. . .). The incremental design is naturally offered to the system architect who can extend easily an existing model by creating new views.

Properties of views can be specified with Bounded Linear Temporal Logic (BLTL) that allows to reason about finite execution. To reason about sequences of view, we propose Adaptive BLTL (A-BLTL) that is an extension of BLTL with an adaptive operator to reason about the dynamic change of views. We also show that the formalism extend to contracts that permit to reason about both assumptions and guarantees of the system.

Consider the system described in Figure 1. This system is composed by three different views linked by adaptive transitions (represented by dashed arrows). Each view contains some system components in a particular topology denoting a configuration of the SAS. Each local property p_1, p_2, p_3 is attached to one or more views. There is also global property Φ. The SAS is initially designed by View 1 and View 2 and the black dashed arrows. Properties p_1, p_2 are validated for the corresponding views and Φ is validated against this complete initial version of the system. To fit with the upcoming settings of the system, the system architect updates the model by adding View 3 with new adaptive transitions (in grey). This requires to only validate p_1 and p_3 against View 3 and to validate again the global property Φ against the system including all the three views.

We propose a new Statistical Model Checking (SMC) [22,20] algorithm to compute the probability for a SAS to satisfy an A-BLTL property. SMC can be seen as a trade-off between testing and formal verification. The core idea of SMC is to generate a number of *simulations* of the system and verify whether they satisfy a given property expressed in temporal logics, which can be done by using *runtime verification approaches* [12]. The results are then used together with algorithms from the statistical area in order to decide whether the system satisfies the property with some probability. One of the key points to implement an SMC algorithm is the ability to bound a priori the size of the simulation, which is an important issue. Indeed, although the SAS can only spend a finite amount of time in a given view, the time bound is usually unknown and simulation cannot be bounded. To overcome the problem, we expand on the work of Clarke [21] and consider a combination of SMC and model checking algorithm for untimed systems.

As a second contribution, we propose high-level formalisms to represent both SAS and A-BLTL/contracts. The formalism used to specify SAS relies on an extension of the Reactive Module Language (RML) used by the popular Prism toolset [16]. Properties are represented with an extension of the Goal and Contract Specification Language (GCSL) [2] defined in the DANSE IP project [11]. This language offers English-based pattern to reason about timed properties without having to learn complex mathematics inherent to any logic.

Finally, as a last contribution, we have implemented our work in Plasma-Lab [5] – a new powerful SMC model checker. The implementation has been tested on a realistic case study defined with industry partners of DANSE.

2 Modeling Stochastic Adaptive Systems

In this section, we present the formal model used to encode behaviors of adaptive systems. In Section 2.1, we introduce Markov chains (MC) to represent individual components of a view. Then, in Section 2.2, we show how to describe views as well as relations between views, i.e., adaptive systems.

2.1 Discrete and Continuous Time Markov Chains

Definition 1. *A (labelled)* transition system *is a tuple $\mathcal{T} = (Q, \overline{q}, \Sigma, \rightarrow, AP, L)$ where Q is a set of states, $\overline{q} \in Q$ is the initial state, Σ is a finite set of actions, $\rightarrow : Q \times \Sigma \times Q$ is the transition relation, AP is a set of atomic propositions, and $L : Q \rightarrow 2^{AP}$ is a state labelling function that assigns to each state the set of propositions that are valid in the state.*

We denote by $q \xrightarrow{a} q'$ the transition $(q, a, q') \in \rightarrow$. A *trace* is a finite or infinite alternating sequence of states and time stamps $\rho = t_0 q_0 t_1 q_1 t_2 q_2 \ldots$, such that $\forall i. \exists a_i \in \Sigma. q_i \xrightarrow{a_i} q_{i+1}$. Time stamps measure the cumulative time elapsed from a time origin. In discrete time models delays are integer values (i.e., $t_0 = 0$, $t_1 = 1$, $t_2 = 2$) and therefore they can be omitted. In continuous time models they are real values. We denote by $|\rho|$ the length of a trace ρ. If ρ is infinite then $|\rho| = \infty$. A trace is initial if $q_0 = \overline{q}$ and $t_0 = 0$. We denote by $\mathsf{trace}_n(\mathcal{T})$ (resp. $\mathsf{trace}(\mathcal{T})$) the set of all initial traces of length n (resp. infinite traces) in $\mathcal{T}$. Let $0 \leq i \leq |\rho|$, we denote $\rho_{|i} = t_0 q_0 t_1 q_1 \ldots t_{i-1} q_{i-1}$ the finite prefix of ρ of length i, $\rho^{|i} = t_i q_i t_{i+1} q_{i+1} \ldots$ the suffix of ρ that starts at position i, and $\rho[i] = q_i$ the state at position i.

We now extend transition systems with probabilities to represent uncertainty of behaviors or of material on which the system is running. We present two semantics, either with discrete or continuous time, that are both compatible with our setting. A discrete time Markov chain (DTMC) is a state-transition system in which each transition is labelled by a probability $\mathbf{P}(s, s')$ to take the transition from state s to state s'.

Definition 2. *A (labelled) DTMC is a tuple $\mathcal{D} = (Q, \overline{q}, \Sigma, \rightarrow, AP, L, \mathbf{P})$ where:*

- $(Q, \overline{q}, \Sigma, \rightarrow, AP, L)$ *is a labelled transition system,*
- $\mathbf{P} : Q \times Q \rightarrow [0,1]$ *is a* transition probability matrix*, such that* $\sum_{q' \in Q} \mathbf{P}(q, q') = 1$ *for all* $q \in Q$*,*
- $\rightarrow$ *is such that* $q \xrightarrow{a} q'$ *iff* $\mathbf{P}(q, q') > 0$*, and for each state* q *there is at most one action* $a \in \Sigma$ *such that* $q \xrightarrow{a} q'$ *for some* q'*.*

In continuous time Markov chains (CTMCs) transitions are given a rate. The sum of rates of all enabled transitions specifies an exponential distribution that determines a real value for the time spent in the current state. The ratio of the rates then specifies which discrete transition is chosen.

Definition 3. *A (labelled) CTMC is a tuple* $\mathcal{C} = (Q, \overline{q}, \Sigma, \rightarrow, AP, L, \mathbf{R})$ *where:*

- $(Q, \overline{q}, \Sigma, \rightarrow, AP, L)$ *is a labelled transition system,*
- $\mathbf{R} : Q \times Q \rightarrow \mathbb{R}_{\geq 0}$ *is a* transition rate matrix*,*
- $\rightarrow$ *is such that* $q \xrightarrow{a} q'$ *iff* $\mathbf{R}(q, q') > 0$*, and there is a unique* $a \in \Sigma$ *such that* $q \xrightarrow{a} q'$*.*

In our setting, a view of a system is represented by the combination of several components. We can compute the parallel composition $C_1 \parallel C_2$ of two DTMCs (resp. CTMCs) defined over the same alphabet Σ. Let $(Q_1, \overline{q}_1, \Sigma, \rightarrow_1, AP_1, L_1)$ and $(Q_2, \overline{q}_2, \Sigma, \rightarrow_2, AP_2, L_2)$ be the two underlying transition systems. We first compute their parallel composition, which is a labelled transition system $(Q, \overline{q}, \Sigma, \rightarrow, AP, L)$, where $Q = Q_1 \times Q_2$, $\overline{q} = (\overline{q}_1, \overline{q}_2)$, $AP = AP_1 \cup AP_2$, $L(q) = L_1(q_1) \cup L_2(q_2)$ and the transition relation $\rightarrow$ is defined according to the following rule:

$$\frac{q_1 \xrightarrow{a}_1 q_1' \quad q_2 \xrightarrow{a}_2 q_2'}{(q_1, q_2) \xrightarrow{a} (q_1', q_2')} \tag{1}$$

Then, in case of DTMCs, the new transition probability matrix is such that $\mathbf{P}((q_1, q_2), (q_1', q_2')) = \mathbf{P}(q_1, q_1') * \mathbf{P}(q_2, q_2')$, and in case of CTMCs the new transition rate matrix is such that $\mathbf{R}((q_1, q_2), (q_1', q_2')) = \mathbf{R}(q_1, q_1') * \mathbf{R}(q_2, q_2')$. DTMCs with different alphabets can also be composed and they synchronize on common actions. However, if both DTMCs can perform a non synchronized action, a uniform distribution is applied to resolve the non determinism. In case of CTMCs, the two actions are in concurrence, such that if $q_1 \xrightarrow{a}_1 q_1'$ with $a \notin \Sigma_2$, then $(q_1, q_2) \xrightarrow{a} (q_1', q_2)$ and $\mathbf{R}((q_1, q_2), (q_1', q_2)) = \mathbf{R}(q_1, q_1')$. In what follows, we denote by $Sys = C_1 \parallel C_2 \parallel \cdots \parallel C_n$ the DTMC (resp. CTMC) that results from the composition of the components $C_1, C_2, \ldots, C_n$.

2.2 Stochastic Adaptive Systems (SAS)

An adaptive system consists in several successive views. It starts in an initial view that evolves until it reaches a state in which an adaptation is possible. This adaptation consists in a view change that depends on a probability distribution that represents uncertainty of the environment.

Definition 4. *A SAS is a tuple* $(\Delta, \Gamma, S, \overline{sys}, \rightsquigarrow)$ *where:*

- $\Delta = \{C_1, C_2, \ldots, C_n\}$ *is a set of DTMCs (resp. CTMCs) that are the components of the SAS.*
- Γ *is the set of views of the SAS, such that each view is a stochastic system obtained from the parallel composition some components from* Δ.
- $\overline{sys} \in \Gamma$ *is the initial view.*
- S *is the set of states of the SAS.* S *is the union of the states of each view in* Γ*, i.e., for each state* $s \in S$ *there exists* $\{C_1, C_2, \ldots, C_k\} \subseteq \Delta$ *such that* $s \in Q_1 \times Q_2 \times \cdots \times Q_k$ *(where* $\forall i, 1 \le i \le k$*,* Q_i *is the set of states of* C_i*).*
- $\rightsquigarrow \subseteq S \times [0,1]^S$ *is a set of adaptive transitions.*

Observe that the number of components per state may vary. This is due to the fact that different views may have different components. Observe also that it is easy to add new views to an existing adaptive system without having to re-specify the entire set of views. An element $(s, \mathbf{p}) \in \rightsquigarrow$ consists in a state s from a view $sys \in \Gamma$ and a probability distribution $\mathbf{p}$ over the states in S. When $s \neq s'$, we denote $s \rightsquigarrow s'$ if there exists $\mathbf{p}$ such that $(s, \mathbf{p}) \in \rightsquigarrow$ and $\mathbf{p}(s') > 0$, which means that state s can be adapted into state s' with probability $\mathbf{p}(s')$.

A trace ρ in a SAS is either a finite combination of n traces $\rho = \rho_0\rho_1 \ldots \rho_n$, such that for all $0 \le i \le n-1$, $\rho_i = t_{0i}s_{0i}t_{1i}s_{1i} \ldots t_{li}s_{li}$ is a finite trace of $sys_i \in \Gamma$, and $s_{li} \rightsquigarrow s_{0i+1}$, and $t_{0i+1} = 0$, and ρ_n is a finite or infinite trace $sys_n \in \Gamma$. Otherwise ρ may be an infinite combination of finite traces $\rho = \rho_0\rho_1 \ldots$ that satisfy for all i the same constraints.

3 A Logic for SAS Properties

3.1 Probabilistic Adaptive Bounded Linear Temporal Logic

We consider quantitative verification of dynamic properties that are expressed via a quantitative extension of the Adaptive Linear Temporal Logic (A-LTL) proposed in [23]. Our logic, which we call Adaptive Bounded Linear Temporal Logic (A-BLTL), relies on an extension of Bounded Linear Temporal Logic (BLTL) combined with an adaptive operator. Although the logic is not strictly more expressive than BLTL, it is more suitable to describe properties of individual views, as well as global properties of the adaptive system, and it allows to develop specific algorithms for these properties. In the last part of the section, we also show how one can define contracts on such logic, where a contract [17] is a pair of assumptions/guarantees that must be satisfied by the system.

We first introduce BLTL, a logic used to express properties on individual views. In BLTL, formulas are built by using the standard Boolean connectors $\wedge$, $\vee$, $\implies$, $\neg$, and the temporal operators G, F, X, U borrowed from Linear Temporal Logic (LTL). The main difference between BLTL and classical LTL is that each temporal modality is indexed by a bound k that defines the length of the run on which the formula must hold. The semantics of a BLTL formula is defined in Table 1 for finite executions of CTMC/DTMC $\rho = t_0s_0t_1s_1t_2s_2 \ldots$,

Table 1. Semantics of BLTL

$\rho \models X_{\leq k}\Phi$	$\equiv \exists i,\ i = max\{j \mid t_0 \leq t_j \leq t_0 + k\}$ and $\rho^{\mid i} \models \Phi$		
$\rho \models \Phi_1\ U_{\leq k}\Phi_2$	$\equiv \exists i,\ t_0 \leq t_i \leq t_0 + k$ and $\rho^{\mid i} \models \Phi_2$ and $\forall j, 0 \leq j \leq i,\ \rho^{\mid j} \models \Phi_1$		
$\rho \models \Phi_1 \wedge \Phi_2$	$\equiv \rho \models \Phi_1$ and $\rho \models \Phi_2$	$\rho \models \neg\Phi$	$\equiv \rho \not\models \Phi$
$\rho \models P$	$\equiv P \in L(s_0)$	$\rho \models true$	$\rho \not\models false$

with $|\rho| \geq k$. If $|\rho| < k$, it is extended by duplicating the last state enough times. In case formulas are nested, the value of k adapts incrementally.

We now generalize BLTL to adaptive systems. For doing so, we introduce an adaptive operator in the spirit of [24]. The new logic A-BLTL is an extension of BLTL with an adaptive operator $\Phi \overset{\Omega}{\Rightarrow}_{\leq k} \Psi$, where Φ is a BLTL formula, Ψ is an A-BLTL formula, Ω is a predicate over the states of different views of the SAS, and k is a time bound that limits the execution time of the adaptive transition. We will also consider unbounded versions of the adaptive operator.

Definition 5 (A-BLTL semantics). *Let $\Phi \overset{\Omega}{\Rightarrow}_{\leq k} \Psi$ be an A-BLTL formula and $\rho = t_0 s_0 t_1 s_1 t_2 s_2 \ldots$ be an execution of the SAS:*

$$\rho \models \Phi \overset{\Omega}{\Rightarrow}_{\leq k} \Psi \equiv \exists i,\ i = min\{j \mid t_0 \leq t_{j-1} \leq t_0 + k \wedge \rho_{\mid j} \models \Phi \wedge s_{j-1} \rightsquigarrow s_j \wedge \Omega(s_{j-1}, s_j)\} \wedge \rho^{\mid i} \models \Psi \quad (2)$$

The property is unbounded if $k = \infty$, and in that case we write $\Phi \overset{\Omega}{\Rightarrow} \Psi$.

According to Definition 5, an execution ρ satisfies an adaptive formula $\Phi \overset{\Omega}{\Rightarrow}_{\leq k} \Psi$ if and only if there exists a minimal prefix of ρ that satisfies Φ and reaches a state s_{j-1}, such that $\Omega(s_{j-1}, s_j)$ is satisfied, and such that the suffix of ρ from state s_j satisfies Ψ. Therefore to satisfy this formula it is necessary to observe an adaptation compatible with Ω. We relax this constraint by introducing a new operator $\Phi \xrightarrow{\Omega}_{\leq k} \Psi$, for which an adaptation is not necessary but triggers a check of Ψ when it happens. It is equivalent to the following formula: $\Phi \xrightarrow{\Omega}_{\leq k} \Psi \equiv (\Phi \overset{\Omega}{\Rightarrow}_{\leq k} true) \implies (\Phi \overset{\Omega}{\Rightarrow}_{\leq k} \Psi)$.

We finally introduce stochastic contracts, that are used to reason about both the adaptive system and its environment via assumptions and guarantees.

Definition 6 (Contracts for SAS). *A* contract *is defined as a pair (A, G), where A and G are respectively called the Assumption and the Guarantee. A SAS $\mathcal{M}$ satisfies the contract (A, G) iff $\forall \rho,\ \rho \models A \Rightarrow \rho \models G$, where ρ is a trace of $\mathcal{M}$ and $\rho \models A$ (resp. G) means the trace ρ satisfies the assumption A (resp. the guarantee G). In that case we write $\mathcal{M} \models (A, G)$.*

3.2 Verifying SAS Properties Using SMC

SMC [22,20] is an alternative to model checking [3,9] that employs Monte Carlo methods to avoid the state explosion problem. SMC estimates the probability that a system satisfies a property using a number of statistically independent simulation traces of an executable model. By using results from the statistic area, SMC decides whether the system satisfies the property with some degree of confidence, and therefore it avoids an exhaustive exploration of the state-space of the model that generally does not scale up. It has already been successfully experimented in biology area [10,15,16], software engineering [8] as well as industrial area like aeronautics [4].

The basic algorithm used in SMC is the Monte Carlo algorithm. This algorithm estimates the probability that a system Sys satisfies a BLTL property P by checking P against a set of N random executions of SyS. The estimation $\hat{p}$ is given by $\hat{p} = \frac{\sum_1^N f(\rho_i)}{N}$ where $f(\rho_i) = 1$ if $\rho_i \models P$, 0 otherwise.

Using the formal semantics of BLTL, each execution trace ρ_i is monitored in order to check if P is satisfied or not. The accuracy of the estimation increases with the number of monitored simulations. This accuracy can be controlled thanks to the Chernoff-Hoeffding bound [14]. It relates N to δ and ε, that are respectively the confidence and the error bound of $\hat{p}$:

$$Pr(|p - \hat{p}| < \varepsilon) \geq 1 - \delta \quad \text{if} \quad N \geq \frac{ln(\frac{2}{\delta})}{2\varepsilon^2} \tag{3}$$

According to this relation, the user is able to trade off analysis time in return for accuracy of the result using the parameters δ and ε.

Knowing that there exists techniques to monitor BLTL properties [13], the model checking of A-BLTL is rather evident. Given an A-BLTL property $\Phi_1 \overset{\Omega}{\Rightarrow}_{\leq k} \Phi_2$, the monitor will first check whether the run satisfies Φ_1 using classical runtime verification techniques. If no, then the property is not satisfied. If yes, then one checks whether there is a pair of two successive states between t_0 and $t_0 + k$ that satisfies Ω. The latter is done by parsing the run. If this pair does not exist, then the property is not satisfied. Else we start a new monitor from the suffix of the run starting in the second state of the pair in order to verify Φ_2.

3.3 Verifying Unbounded SAS Properties Using SMC

We propose a method inspired by [21] to check unbounded A-BLTL properties. The principle is to combine a reachability analysis by model checking the underlying finite-state machine, with a statistical analysis of the stochastic model using the algorithms introduced previously.

We consider an A-BLTL property $\Phi \overset{\Omega}{\Rightarrow} \Psi$, where Φ and Ψ are BLTL formulas. We first consider the reachability problem with objective $G = \{s \mid \exists s'.s \rightsquigarrow s' \wedge \Omega(s, s')\}$. The preliminary to the statistical analysis is to compute the set $Sat(Reach(G))$, that is all the states of the SAS that may eventually reach a state in G. This can be computed using classical model checking algorithms for finite-state machines. Only the underlying automata of the DTMCs or CTMCs of the SAS is used for this analysis. As stochastic quantities are ignored, efficient symbolic techniques exist to speed up this process [6].

```
 1: procedure CHECK($\Phi \overset{\Omega}{\Rightarrow} \Psi, \rho$)
 2:     if $\neg$CHECK($\Phi, \rho$) then return false
 3:     end if
 4:     $i \leftarrow 0$, $prec \leftarrow null$, $curr \leftarrow null$
 5:     while $i < |\rho|$ do
 6:         $prec \leftarrow curr$, $curr \leftarrow \rho[i]$
 7:         if $curr \notin Sat(Reach(G))$ then return false
 8:         end if
 9:         if $\Omega(prec, curr)$ then return CHECK($\Psi, \rho^{|i}$)
10:         end if
11:         $i \leftarrow i + 1$
12:     end while
13: end procedure
```

Fig. 2. Algorithm to monitor unbounded A-BLTL properties

Once this preliminary computation is performed, the CHECK algorithm from Figure 2 is used to monitor the runs of the SAS. The algorithm takes as input the A-BLTL property and a run ρ. The run should be in general infinite as there is no bound on the length of the runs that satisfied an unbounded A-BLTL property. In that case the states would be generated on-the-fly. The algorithm returns true or false, whether the run satisfied the property. We also denote CHECK(Φ, ρ) as the monitoring of the BLTL property Φ. Then the first step on line 2 is to monitor Φ on the run ρ. If the result is false then the property $\Phi \overset{\Omega}{\Rightarrow} \Psi$ is not satisfied. Otherwise the algorithm searches through ρ for two states *prec* and *curr* such that $\Omega(prec, curr)$ is true. This is possible if *curr* belongs to the precomputed set $Sat(Reach(G))$. If Ω is satisfied the last step on line 10 is to monitor Ψ from the current position in ρ.

For homogeneous Markov chains (with constant probability matrices, as it is the assumption in this paper), the algorithm almost surely (with probability 1) terminates, since it either reaches a state where Ω is unreachable, or the probability to reach two states that satisfy Ω is not null. It can be iterated to check sequences of adaptive operators, that is to say properties where Ψ is also an unbounded A-BLTL.

4 A Software Engineering Point of View

In this section, we propose high level formalisms to specify both adaptive systems and their properties. Then, we define semantics of those formalisms by exploiting the definitions introduced in the previous sections. This gives us a free verification technology for them. The situation is illustrated in Figure 3.

4.1 Adaptive RML Systems as a High Level Formalism for SAS

We represent adaptive systems with Adaptive Reactive Module Language (A-RML), an extension of the Reactive Module Languages (RML) used by the

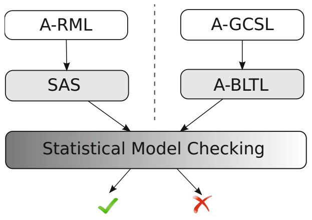


Fig. 3. SAS verification flow

PRISM toolset [16]. Due to space limit, the syntax common to RML and A-RML is only briefly described here [1].

The RML language is based on the synchronisation of a set of modules defined by the user. A module is declared as a DTMC or CTMC, i.e., some local variables with a set of guarded commands. Each command has a set of actions of the form λ_i:$\mathtt{a}_i$ where λ_i is the probability (or the rate) to execute $\mathtt{a}_i$. A-RML extends RML such that each module can have some parameters in order to define its initial state.

```
module MOD_NAME(<Parameters>)
   <local_vars>
      ...
   [chan] g_k -> (λ_0:a_0) + ... + (λ_n:a_n);
      ...
endmodule
```

The optional channel identifiers prefixing commands are used to strongly synchronise the different modules of a RML system. A module is synchronised over the channel `chan` if it has some commands prefixed by `chan`. We say a command is independent if it has no channel identifier.

In A-RML a system is a set of modules and global variables. The modules synchronise on common channels such that the system commands are the independent commands of each module and the synchronised commands. A command synchronised on `chan` forces all the modules that synchronised over `chan` to simultaneously execute one of their enabled commands prefixed by `chan`. If one module is not ready, *i.e.* it has no enabled commands for `chan`, the system has no enabled command over `chan`. Similarly to a module, if the system reaches a state with a non-deterministic choice, it is solved by a stochastic behaviour based on a uniform distribution. This solution allows to execute the A-RML system in accordance with the DTMC/CTMC models.

```
system SYS_NAME(<Parameters>)
   <global_variables>
   ...
   <module_declarations>
   ...
endsystem
```

[1] The full syntax can be found at `http://project.inria.fr/plasma-lab/`

An adaptive system consists in a set of different views, each represented by an A-RML system, and a list of adaptations represented by adaptive commands. The `adaptive` environment is used to specify which one is the initial view and what adaptations are possible.

```
adaptive
   init at SYS_NAME(<Initial values>)
   ...
   { SYS_NAME | gk } -> λ0:{a0} + ... + λn:{an};
   ...
endadaptive
```

An adaptive command is similar to module command. It has a guard $\mathtt{g}_k$ that applies to the current view `SYS_NAME`, and a set of actions λ_i:$\mathtt{a}_i$ where λ_i is the probability (or the rate) to execute action $\mathtt{a}_i$ = `SYS_NAME'`$(\mathtt{e}_0, \ldots, \mathtt{e}_m)$. This action defines the next view `SYS_NAME'` after performing the adaptation. This view is determined according to the states of the previous view by setting the parameters of `SYS_NAME'` with the expressions $\mathtt{e}_0, \ldots, \mathtt{e}_m$ evaluated over `SYS_NAME`. The execution of the adaptive command is done in accordance with the SAS semantics.

Theorem 1. *The semantics of A-RML can be defined in terms of SAS.*

The proof of the above theorem is a direct consequence of the fact that semantics of RML is definable in terms of composition of MC, and that the definition of an adaptive command can also be represented as a MC.

4.2 A Contract Language for SAS Specification

The Goal and Contract Specification Language (GCSL) was first proposed in [2] to formalise properties of adaptive systems in the scope of the DANSE project. It has a strong semantics based on BLTL but it has a syntax close to the hand written English requirements. Dealing with formal temporal logic is often an issue to formalise correctly the initial English requirements. Most of the time the formalisation frequently contains some mistakes, which is due to the nesting of the temporal operators. The difficulty for correctly specifying properties is enough to make the overall methodology useless.

The GCSL syntax combines a subset of the Object Constraint Language (OCL) [18] (used to define state properties, i.e., Boolean relations between the system components) and English behavioural patterns used to express the evolution of these state properties during the execution of the system. The usage of OCL is illustrated in Example 1.

Example 1. We consider a SAS describing the implementation of an emergency system in a city. The city area is divided as a set of districts where each district may have some equipment to fight against the fire, e.g. some fire stations with fire brigades and fire fighting cars. Each district is also characterised by a risk of fire and the considered damages are mainly related to the population size of each district. The requirement *"Any district cannot have more than 1 fire station, except if all districts have at least 1"* ensures the minimal condition for the equipment distribution in the city. We use syntactic coloration to make the difference between the parts of the language used in the property: the words in red are identifiers from the model, the blue part is from OCL, like collection handling, and the black words are variables:

City.itsDistricts→exists(district | district.**ownedFireStations** $>$ **1) implies**
City.itsDistricts→forAll(district | district.**ownedFireStations** $\geq$ **1)**

GCSL patterns are used to specify temporal properties. In this section we only present a subset of such patterns that is considered to be general enough to specify properties of a large set of industry-examples from the DANSE project. After having read this section, the user shall understand that the set can be easily increased. Each pattern can nest one or more state properties, denoted in the grammar by the non-terminals `<OCL-prop>` and `<arith-rel>`, that respectively denote a state property written in OCL or an arithmetic relation between the identifiers used in the model. The non-terminal `<int>` denotes a finite time interval over which the temporal pattern is applied, and `<N>` is a natural number. The patterns can be used directly or combined with OCL: applying a pattern to a collection of system components defines a behavioural property that is applied to each element of the collection. We present below an excerpt of the complete GCSL grammar available in [2]:

```
<GCSL> ::= <OCL-coll>->forAll(<variable>| <pattern>)
| <OCL-coll>->exists(<variable>| <pattern>)
| <OCL-prop>
| <pattern>

<pattern> ::= whenever [<prop>] occurs [<prop>] holds during following [<int>]
| whenever [<prop>] occurs [<prop>] implies [<prop>] during following [<int>]
| whenever [<prop>] occurs [<prop>] does not occur during following [<int>]
| whenever [<prop>] occurs [<prop>] occurs within [<int>]
| [<prop>] during [<int>] raises [<prop>]
| [<prop>] occurs [<N>] times during [<int>] raises [<prop>]
| [<prop>] occurs at most [<N>] times during [<int>]
| [<prop>] during [<int>] implies [<prop>] during [<int>] then [<prop>] during [<int>]

<prop> ::= <OCL-prop> | <arith-rel>
```

Example 2. Consider the following requirement about the model described in Example 1: *"The fire fighting cars hosted by a fire station shall be used all simultaneously at least once in 6 months"*. This requirement uses both GCSL and OCL patterns:

City.itsFireStations→forAll(fStation | **Whenever** [fStation.**hostedFireFightingCars → exists(**ffCar | ffCar.**isAtFireStation)**] **occurs,** [fStation.**hostedFireFightingCars→forall(**ffCar | ffCar.**isAtFireStation** = **false)**] **occurs within** [**6 months**]**)**

We now propose A-GCSL, a syntax extension for GCSL that can be used to describe adaptive requirements of SAS. A-GCSL extends the GCSL grammar by adding a new pattern that allows to express adaptive relations as done with the two adaptive operators defined in Section 3. The first pattern of `<dyna-spec>` is equivalent to the operator $\overset{\Omega}{\Rightarrow}$ and the second one denotes $\xrightarrow{\Omega}$. Any adaptive requirement has three elements (A, Ω, G) that are called assumption, trigger and guarantee, respectively. The assumption and guarantee are specified in GCSL, whereas the trigger is in OCL. The syntax allows to compose the patterns by specifying the guarantee with an adaptive pattern. For instance, a composed requirement of the form `if` Φ_1 `holds and for all rule that satisfies` Ω `then` (`if` Φ_2 `holds and for all rule that satisfies` Ω' `then` Φ_3 `holds`) `holds` is equivalent to the property $\Phi_1 \overset{\Omega}{\Rightarrow} \Phi_2 \overset{\Omega'}{\Longrightarrow} \Phi_3$. The A-GCSL grammar is the following:

```
<dyna-spec> ::= if [<GCSL>] holds and for all rule that satisfies [<prop>]
                then ( <GCSL> | <dyna-spec> ) holds
| if [<GCSL>] holds then there exists a rule satisfying [<prop>]
                and ( <GCSL> | <dyna-spec> ) holds
```

Example 3. Consider again the system in Example 1 and the following A-GCSL requirement:
if [**City.underFire** $=$ **0**] **holds and for all rule such that rule satisfies** [**City.underFire** $\geq$ **3**] **then** [**City.itsDistricts**$\rightarrow$**forall(**district | district.**decl** $=$ **false** $\Longrightarrow$ **whenever** [district.**decl** $=$ **true**] **occurs,** [district.**fire** $=$ **0**] **occurs within** [**50 hours**] **)**] **holds**

The attribute `underFire` denotes the number of districts in which a fire has been declared. If there are more than three fires in the city, then the fire stations change their usual emergency management into a crisis one. When such management is activated, the firemen have 50 hours to fix the problem. The requirement can be translated in A-GCSL using the following formula:

$$\Phi_6 = \big(underFire = 0\big) \xrightarrow{underFire \geq 3}_{\leq 10000} \bigwedge_{d_i : district} \Big(\neg d_i.decl \implies G_{\leq 10000}\ (d_i.decl \implies F_{\leq 50}\ d_i.fire = 0)\Big)$$

In [2], we have showed that any GCSL pattern can be translated into a BLTL formula. The result extends as follows.

Theorem 2. *Any A-GCSL pattern can be translated into an A-BLTL property.*

This result is an immediate consequence of the definition of the adaptive pattern.

5 Experiments with SAS

Our work has been implemented in a new statistical model checker named Plasma-Lab [5], a platform that includes efficient SMC algorithms, a graphical user interface (GUI) and multiple plugins to support various modelling languages. The tool is written in Java that offers maximum cross-platform compatibility. The GUI allows to create projects and experiments, and it implements a simple and practical mean of distributing simulations using remote clients. Plasma-Lab also provides a library to write new plugins and create custom statistical model checkers based on arbitrary modelling languages. Developers will only need to implement a few simple methods to build a simulator and a logic checker, and then beneficiate from Plasma-Lab SMC algorithms.

Plasma-Lab can be used as a standalone application or be instantiated within other softwares. It has already been incorporated in various performance-critical softwares and embedded hardware platforms. The current plugins allow to simulate biologic models, models written in RML and A-RML, but it has also the capabilities to drive an external engine to perform the simulations, like MATLAB, Scilab, or DESYRE a simulator for adaptive systems developed by Ales [1].

5.1 CAE Model

Together with our industrial partners in the DANSE project, we have developed the Concept Alignment Example (CAE). The CAE is a fictive adaptive system example inspired by real-world Emergency Response data to a city fire. It has been built as a playground to demonstrate new methods and models for the analysis and visualization of adaptive systems designs.

The CAE describes the organization of the firefighting forces. We consider in our study that the city is initially divided into 4 districts, and that the population might increase by adding 2 more districts. Different and even more complex examples can be built using the components of this design.

A fire station is assigned to the districts, but as the fire might spread within the districts, the system can adapt itself by hiring more firemen. We can therefore design a SAS with three views as described in Figure 4.

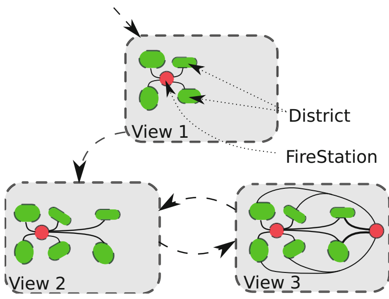


Fig. 4. Components and Views in the CAE model

Adaptive transitions exist between these views to reflect changes in the environment and adaptations of the system. Initially in View 1, the system can switch to View 2 when the population of the city increase. This change models an uncertainty of the environment, and for the purpose of this study we fix its probability to 0.01 Then, if the number of fires becomes greater than 2, the system adapts itself by switching to View 3. If the number of fires eventually reduces and becomes lower than 2, the system might return to View 2. Again, as this change is uncertain, we fix its probability to 0.8.

We design several A-RML models of the system that consist in two types of modules: `District` and `FireStation`, both based on a CTMC semantics. First, we study a model `AbstractCAE` that is an abstract view of the SAS. In this model, the `District` module, presented below, is characterized by a constant parameter `p`, that determines the probability of fire, and by two Boolean variables `decl` and `men`, that respectively defines if a fire has been declared and if the firemen are allocated to the district. The module `fireStation` has one constant parameter $\texttt{distance}_\texttt{i}$ for each module of the system. This parameter determines the probability to react at a fire, such that the greater the distance, the lower the probability. However a fire station can only treat one fire at a time, which is encoded with a Boolean variable `allocated`. The fire stations and the districts synchronize on channels `allocate` and `recover`, that respectively allocate firefighters to the district and bring them back when the fire is treated. The different views are constructed by instantiating and renaming the modules presented above.

```
module District( const int p )
  decl : bool init false;
  men: bool init false;
  [] !decl -> p/1000: (decl'=true);
  [allocate] decl & !men -> (men'=true);
  [recover] decl & men -> 1/p: (decl'=false) & (men'=false);
endmodule
```

We refine this model to better encode the behaviour of the SAS. In this new model `ConcreteCAE` a new variable `fire` of module `District` ranges from 0 to 10 and grades the intensity of the fire. The fire stations can now assign several cars (from 0 to 5) to each districts. Therefore the variables `men` and `allocated` becomes integers.

```
module District( const int p )
  fire : [0..10] init 0;
  decl : bool init false;
  men: [0..5] init 0;
  [] fire=0 -> p/1000: (fire'=1);
  [] fire>0 & fire<10 -> p/((1+men)*100): (fire'=fire+1);
  [] fire>0 & !decl -> (fire*fire)/10: (decl'=true);
  [allocateSt1] decl & fire>0 -> (fire*fire)/10: (men'=men+1);
  [allocateSt2] decl & fire>0 -> (fire*fire)/10: (men'=men+1);
  [] men>0 & fire>0 -> men/10: (fire'=fire-1);
  [recover] decl>0 & fire=0 -> 1000: (men'=0)&(decl'=false);
endmodule
```

From the two models we can consider several subparts composed by one or several views of the SAS. Adaptive commands are used to model the transitions between the different views.

– `AbstractCAE_1` consists in View 1 and 2 from model `AbstractCAE`.
– `AbstractCAE_2` consists in View 2 and 3.
– `AbstractCAE_3` has the same views as `AbstractCAE_2` but is initiated in View 3 instead of View 2.

– ConcreteCAE_1 only consists in View 1 from model ConcreteCAE.
– ConcreteCAE_2 only consists in View 2.
– ConcreteCAE_3 only consists in View 3.
– ConcreteCAE_Full is the full model of ConcreteCAE, with the 3 views and all the adaptive transitions between them.

5.2 Checking Requirements

The requirements are expressed in A-GCSL and translated to A-BLTL. We first check the model AbstractCAE against A-BLTL properties with adaptive operators. Our goal is to verify that the transitions between the different views of the system occurs and satisfy some properties.

The first property, *if* [true] *holds then there exists a rule satisfying* [underfire $\leq$ 1] *and Always* [!maxfire], checks that when the system is in View 1, it eventually switches to View 2 when the number of districts that have declared a fire (underfire) is still lower than 1, and that as a result the system remains safe for a limited time period, i.e., the number of districts that have declared a fire is not maximum (maxfire is false). To check this property we limit the analysis to the model AbstractCAE_1 with only View 1 and View 2. The A-GCSL property is translated in an A-BLTL formula: $\Phi_1 = \texttt{true} \xRightarrow{\texttt{underfire} \leq 1} G_{\leq 1000}\ !\texttt{maxfire}$, and the results in Table 2 show that the probability to satisfy the property is only 50%. This justify the need to add a second fire station, as in View 3.

The second property, *if* [true] *holds then there exists a rule satisfying* [true] *and Always* [!maxfire], checks that from View 2 a second fire station is quickly added, which switches the system to View 3, and that then the system is safe. The property is checked on the model AbstractCAE_2 using the A-BLTL formula : $\Phi_2 = \texttt{true} \xRightarrow{\texttt{true}}_{\leq 100} G_{\leq 10000}\ !\texttt{maxfire}$.

Finally, with the property *if* [true] *holds then there exists a rule satisfying* [true] *and* [true], we check that from View 3 the system eventually returns to View 2. Therefore we use the model AbstractCAE_3 that starts in View 3 and we check the A-BLTL formula $\Phi_3 = \texttt{true} \xRightarrow{\texttt{true}}_{\leq 100} \texttt{true}$.

The AbstractCAE models are simple enough to be able to perform reachability analyses and check the unbounded A-BLTL properties presented above using Algorithm 2. In a second step we consider the models ConcreteCAE to better evaluate the safety of the system. The state spaces of these models contain several millions of states, and therefore, they can only be analyzed by purely SMC algorithms. We verify the two following properties:

– *Always* !maxfire, to check that the maximum of fire intensity of 10 is never reached in any district. This corresponds to $\Phi_4 = G_{\leq 10000}\ !\texttt{maxfire}$.
– *Whenever* [fire > 0] *occurs* [fire = 0] *within* [50 *hours*], to check that a fire in a district is totally extinct within 50 hours. This corresponds to $\Phi_5 = G_{\leq 10000}\big(\texttt{d6.fire} > 0 \implies F_{\leq 50}\ \texttt{d6.fire} = 0\big)$.

These two properties are first checked for each view of the system. The results in Table 2 show that while View 1 and View 3 are surely safe, View 2 is frequently unsafe. But when we check these properties on the complete adaptive model `ConcreteCAE_Full`, with the three views, we can show that the system remains sufficiently safe. It proves that after a change of the environment (the increase of population) the system is able to adapt itself to guaranty its safety.

In the last experiment of Table 2 we check the A-GCSL property presented in Example 3. This bounded adaptive A-GCSL property is checked using the full `ConcreteCAE` model.

We have performed each experiment in Plasma-Lab with a confidence $\delta = 0.01$ and an error bound $\varepsilon = 0.02$. The results in Table 2 give the probabilities estimation and the time needed to perform the computation.

Table 2. Experiments on CAE models

Property	CAE Model	Estimation interval	Consumed Time
Φ_1	`AbstractCAE_1` View 1, View 2	$[0.53, 0.56]$	1351s
Φ_2	`AbstractCAE_2` View 2, View 3	$[0.84, 0.86]$	11s
Φ_3	`AbstractCAE_3` AbstractCAE_2 starting from View 3	$[0.98, 1]$	1363s
Φ_4	`ConcreteCAE_6` 4 dist. 1 sta.	$[0.95, 0.99]$	11s 9s
Φ_4 Φ_5	`ConcreteCAE_2` 6 dist. 1 sta.	$[0.46, 0.5]$ $[0.21, 0.25]$	15s 13s
Φ_4 Φ_5	`ConcreteCAE_3` 6 dist. 2 sta.	$[0.98, 1]$ $[0.98, 1]$	30s 31s
Φ_4 Φ_5	`ConcreteCAE_Full` 4-6 dist. 1-2 sta.	$[0.89, 0.93]$ $[0.82, 0.86]$	25s 42s
Φ_6	`ConcreteCAE_Full` 4-6 dist. 1-2 sta.	$[0.47, 0.51]$	109s

6 Conclusion

This paper presents a new methodology for the rigorous design of stochastic adaptive systems. Our model is general, but the verification procedure can only reason on a finite and known set of views. Our formalism is inspired from [24], where both the stochastic extension and high level formalisms are not considered. In future work, we will extend this approach to purely dynamic systems. Another objective is to extend the work to reason about more complex properties such as energy consumption. Finally, we shall exploit extensions of SMC algorithms such as CUSUM [19] which permits to reason on switches of probability satisfaction. This would allow us to detect emergent behaviors.

References

1. Ales Corp.: Advanced laboratory on embedded systems, `http://www.ales.eu.com/`
2. Arnold, A., Boyer, B., Legay, A.: Contracts and behavioral patterns for systems of systems: The EU IP DANSE approach. In: AiSoS. EPTCS (2013)

3. Baier, C., Katoen, J.P.: Principles of Model Checking. The MIT Press (2008)
4. Basu, A., Bensalem, S., Bozga, M., Delahaye, B., Legay, A.: Statistical abstraction and model-checking of large heterogeneous systems. Int. J. Softw. Tools Technol. Transf. 14(1), 53–72 (2012)
5. Boyer, B., Corre, K., Legay, A., Sedwards, S.: PLASMA-lab: A flexible, distributable statistical model checking library. In: Joshi, K., Siegle, M., Stoelinga, M., D'Argenio, P.R. (eds.) QEST 2013. LNCS, vol. 8054, pp. 160–164. Springer, Heidelberg (2013)
6. Burch, J.R., Clarke, E., McMillan, K.L., Dill, D., Hwang, L.J.: Symbolic model checking: 1020 states and beyond. In: LICS, pp. 428–439 (1990)
7. Cheng, B.H.C., et al.: Software engineering for self-adaptive systems: A research roadmap. In: Cheng, B.H.C., de Lemos, R., Giese, H., Inverardi, P., Magee, J. (eds.) Self-Adaptive Systems. LNCS, vol. 5525, pp. 1–26. Springer, Heidelberg (2009)
8. Clarke, E., Donzé, A., Legay, A.: On simulation-based probabilistic model checking of mixed-analog circuits. Form. Methods Syst. Des. 36(2), 97–113 (2010)
9. Clarke Jr., E.M., Grumberg, O., Peled, D.A.: Model checking. MIT Press, Cambridge (1999)
10. Clarke, E.M., Faeder, J.R., Langmead, C.J., Harris, L.A., Jha, S.K., Legay, A.: Statistical model checking in *bioLab*: Applications to the automated analysis of T-cell receptor signaling pathway. In: Heiner, M., Uhrmacher, A.M. (eds.) CMSB 2008. LNCS (LNBI), vol. 5307, pp. 231–250. Springer, Heidelberg (2008)
11. DANSE: Designing for adaptability and evolution in sos engineering (December 2013), `https://www.danse-ip.eu/home/`
12. Havelund, K., Rosu, G.: Preface. ENTCS 70(4), 201–202 (2002), Runtime Verification
13. Havelund, K., Roşu, G.: Synthesizing monitors for safety properties. In: Katoen, J.-P., Stevens, P. (eds.) TACAS 2002. LNCS, vol. 2280, pp. 342–356. Springer, Heidelberg (2002)
14. Hoeffding, W.: Probability inequalities for sums of bounded random variables. Journal American Statistical Association 58(301), 13–30 (1963)
15. Jha, S.K., Clarke, E.M., Langmead, C.J., Legay, A., Platzer, A., Zuliani, P.: A bayesian approach to model checking biological systems. In: Degano, P., Gorrieri, R. (eds.) CMSB 2009. LNCS, vol. 5688, pp. 218–234. Springer, Heidelberg (2009)
16. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 585–591. Springer, Heidelberg (2011)
17. Meyer, B.: Applying "design by contract". Computer 25(10), 40–51 (1992)
18. OMG: Ocl v2.2 (February 2010), `http://www.omg.org/spec/OCL/2.2/`
19. Page, E.S.: Continuous inspection schemes. Biometrika 41(1/2), 100–115 (1954)
20. Sen, K., Viswanathan, M., Agha, G.: On statistical model checking of stochastic systems. In: Etessami, K., Rajamani, S.K. (eds.) CAV 2005. LNCS, vol. 3576, pp. 266–280. Springer, Heidelberg (2005)
21. Younes, H.L.S., Clarke, E.M., Zuliani, P.: Statistical verification of probabilistic properties with unbounded until. In: Davies, J., Silva, L., Simão, A. (eds.) SBMF 2010. LNCS, vol. 6527, pp. 144–160. Springer, Heidelberg (2011)
22. Younes, S., Clarke, E.M., Gordon, G.J., Schneider, J.G.: Verification and planning for stochastic processes with asynchronous events. Tech. rep. (2005)
23. Zhang, J., Cheng, B.H.C.: Model-based development of dynamically adaptive software. In: ICSE. ACM (2006)
24. Zhang, J., Cheng, B.H.: Using temporal logic to specify adaptive program semantics. Journal of Systems and Software 79(10), 1361–1369 (2006)

A Review of Statistical Model Checking Pitfalls on Real-Time Stochastic Models*

Dimitri Bohlender[1], Harold Bruintjes[1], Sebastian Junges[1], Jens Katelaan[1], Viet Yen Nguyen[1,2], and Thomas Noll[1]

[1] Software Modeling and Verification Group, RWTH Aachen University, Germany
[2] Fraunhofer IESE, Germany

Abstract. Statistical model checking (SMC) is a technique inspired by Monte-Carlo simulation for verifying time-bounded temporal logical properties. SMC originally focused on fully stochastic models such as Markov chains, but its scope has recently been extended to cover formalisms that mix functional real-time aspects, concurrency and non-determinism. We show by various examples using the tools UPPAAL-SMC and Modes that combining the stochastic interpretation of such models with SMC algorithms is extremely subtle. This may yield significant discrepancies in the analysis results. As these subtleties are not so obvious to the end-user, we present five semantic caveats and give a classification scheme for SMC algorithms. We argue that caution is needed and believe that the caveats and classification scheme in this paper serve as a guiding reference for thoroughly understanding them.

1 Introduction

Statistical model checking (SMC) techniques [21] have been proposed for overcoming many challenges in devising tractable numerical methods for probabilistic models. Inspired by Monte-Carlo simulation, SMC techniques simulate a random system for a particular number of times, such that statistical evidence is built up for deciding whether a property holds on the model. This method is lightweight and easily parallelizable. Drawn by their tractability, research groups have subsequently adapted them to more-expressive classes of models [17]. A particular class, which we refer to as *real-time stochastic models*, are characterized by intertwining concurrency, non-deterministic choice, real-time and probabilistic aspects into a single formalism. We consider these models as partially stochastic, as they have no common stochastic interpretation on their non-probabilistic language elements, such as concurrency. Yet, their potential applications are clear. For example, real-time stochastic models could be used for expressing safety-critical embedded systems, which are constructed for validating RAMS (reliability, availability, maintainability and safety) requirements. The probabilistic

* This work was partially supported by ESA/ESTEC (contract no. 4000107221 (HASDEL)) and the EU (project reference 318772 (D-MILS) and project reference 318490 (SENSATION)).

features are often used for expressing uncertain faulty/erroneous behavior, like for example bit-flips, sensor glitches or simply the complete failure of a component. Real-time aspects are generally applied for expressing time-critical behavior, such as the nominal functional operation, as well as time-constrained behavior, like failure recovery mechanisms. On top of that, concurrency, both in the system and its environment, is omni-present to facilitate compositionality. Non-deterministic choice can be used to express the various potential implementations in a single model.

At this moment of writing, there are two prominent publicly available SMC tools that handle a real-time stochastic modeling formalism, namely UPPAAL-SMC [11] and Modes [7]. While using and comparing them, we noticed unintuitive discrepancies in the computed probabilities that could not be easily dismissed as bugs. After launching a full investigation, we found that there are deeply-rooted semantic caveats, as well as algorithmic design aspects that have a major impact on the computed probabilities. They emanate from mixing concurrency *and* non-deterministic choice *and* real-time *and* probabilistic aspects. This paper reports on the lessons we learned by the following contributions:

- The identification of five caveats and their interactions in Sections 3 and 4.
- A classification scheme for SMC algorithms, discussed in Section 5.
- Application and practical evaluation of the above with UPPAAL-SMC [11] and Modes [7] in Section 6.

Note that these lessons strictly apply to real-time stochastic formalisms that mix all four aspects. Model formalisms that mix two or three of those aspects tend to have a fully probabilistic semantics. Hence in those cases, such as Interactive Markov Chains [15] or Timed Automata, our learned lessons do not apply. The intent of our lessons is not to judge the use of SMC techniques on real-time stochastic models. Rather, we believe this paper provides a reference for users and developers to make an informed and deliberate decision on which SMC technique suits their purposes.

2 Preliminaries, Notations and Related Work

The results in this paper are scoped to real-time stochastic models, which we will loosely characterize in this section.

Real-Time Stochastic Models

Real-time stochastic models have a concurrent, a real-time and a probabilistic dimension. Particular examples of them are (networks of) Priced Timed Automata (PTA) as used by UPPAAL-SMC [11] and Stochastic Timed Automata (STA) as used by Modes [7]. Both models share a particular set of ingredients. The focus of this paper does not lie on the complete semantics of either model per se, but rather addresses issues related to a wider and more general class of real-time stochastic automata. Therefore, the semantics only provide a basis for the further discussion in this paper, and do not form a single consistent model. We refer the reader to the original work on PTA and STA for their complete semantics.

Timed Processes. We first introduce a *timed process*, defined as a tuple $P = \langle L, l_0, I, T, C, A\rangle$, with L the set of locations, l_0 the initial location in L, C the set of real-valued clocks, A the set of actions including internal action τ, and

- $I : L \to Expr$ is the function that assigns to each location an invariant expression in $Expr$, which is a Boolean expression over the clocks in C. They restrict the residence time in that location based on the valuation of the clocks.
- $T \subseteq L \times A \times Expr \times 2^C \times L$ denotes the set of discrete transitions, with $Expr$ being the set of Boolean guard expressions over the clocks C.

A transition $\langle l_s, \alpha, g, X, l_t\rangle \in T$ allows the system to move from location l_s to l_t, executing action α. The transition is only enabled if the expression g evaluates to true in the current state. Upon execution of the transition, the clocks in set X are reset to zero.

The continuous state space of a timed process is defined as a set of tuples, consisting of the process' current location and valuation of the clocks. For process P at initial location p_0, with a single clock c_p assigned the value zero, the state is defined as $\langle p_0, (c_p, 0.0)\rangle$. For a process with a single clock we abbreviate this by $\langle p_0, 0.0\rangle$. Transitions between states either delay or are discrete. *Delay* transitions progress clock values, e.g. $\langle p_0, 0.0\rangle \xrightarrow{4.5} \langle p_0, 4.5\rangle$. *Discrete* transitions are caused by a transition in a process' transition relation, e.g. $\langle p_0, 4.5\rangle \xrightarrow{\alpha} \langle p_1, 4.5\rangle$. A *path* (or sometimes also called *trace* or *run*) is a sequence of alternating delay and discrete transitions, e.g. $\langle p_1, 6.5\rangle \xrightarrow{2} \langle p_1, 8.5\rangle \xrightarrow{\alpha} \langle p_2, 8.5\rangle \xrightarrow{1} \ldots$.

For expressing concurrency, we define a *network* of communicating processes (or sometimes called a network of automata). It consists of one or more timed processes composed together using a synchronization operator, denoted by $\|$. We define the state space of such a model as the cross-product of the state space of each individual process. In such a model, discrete transitions occur either concurrently, synchronizing on the shared communication *alphabet* of all the sets A of each process, or independently. When multiple independent transitions are enabled, a race condition occurs that is resolved by means of a *scheduler*. Timed transitions globally increase all clocks at the same rate.

Probabilistic Semantics. In the previous section we described the syntactical concepts for expressing a network of timed processes. We further extend upon this by introducing the possible ways of giving them a probabilistic semantics through inferring probability distributions and assuming a strategy for interpreting concurrency probabilistically.

Two types of stochastic behavior can be specified, namely for discrete transitions and delay transitions. In the case of a set of non-deterministic discrete transitions T, a discrete probability distribution $Distr(T)$ can be defined over these transitions. This distribution specifies for each transition the probability of taking that transition, see e.g. [12,19]. Non-deterministic choice on its own, however, has no probabilistic semantics and thus typically one assumes one while simulating, like a uniform distribution. Additionally, there are formalisms that capture a discrete probabilistic distribution over successor locations. However w.l.o.g. they can be represented by a set of transitions such that we only have to restrict ourselves to models where transitions only have a single successor location. For delay transitions, a probability density function $\eta(l)$ can be specified over the possible delay times in the current location l. As we reason over a dense notion of time, this has to be a continuous distribution. This distribution can be explicitly specified by the user, or it is inferred from invariants or annotations on outgoing transitions like guards. Both the inference of a delay distribution and the choice for a discrete branching distribution come with caveats, which are discussed in Section 3.

In the face of concurrency, a statistical model checker employs a scheduler that selects the process to fire the next transition (or processes in the case of synchronization). This can be a scheduler that assigns a uniform distribution over the processes with the shortest waiting time, such as in UPPAAL-SMC. Other approaches exist as well. This also induces a caveat that is discussed in Section 3.

Recent work of [7] describes networks of Price Timed Automata, extending PTA [2,5] with exit rates for locations, as well as concurrency and probabilistic semantics for SMC. Like Timed Automata [1], they support the specification of clock variables, with location invariants and transition guards. It is the formalism implemented by the UPPAAL-SMC tool.

The work by [8] describes the MoDeST language that provides constructs for expressing the concurrency, probabilistically distributed waiting times, probabilistic branching and non-deterministic branching aspects described above. A set of MoDeST processes are mapped upon a *stochastic timed automaton (STA)*, which can be viewed as a generalization of Timed Automata and various Markov processes, e.g. CTMDPs [15]. It is the formalism implemented by the Modes tool.

Statistical Model Checking

Statistical model checking techniques [17] build upon and extend Monte-Carlo simulation techniques for verifying temporal bounded properties. These techniques rely on the following components: A path generator (sometimes also referred to as discrete event simulator), a property checker that decides whether path generation should continue or not, and an algorithm that decides if more paths need to be generated. The path generator simulates the behavior of the model by repeating discrete or delay transitions. Path generation continues until the property can be decided to hold or not, or when a boundary condition has been met. This is referred to as the *termination condition*. The Monte-Carlo method approximates the probability of a property by generating a large number of random samples. By repeatedly sampling random paths (also called traces or simulations) from the automaton, statistical evidence is built up for disproving or proving the property of interest. Essentially, each generated path satisfies or dissatisfies the property. The outcomes of the generated paths are used to statistically decide whether additional paths are required and what the estimated probability of the property is. We refer the reader to [17] for an overview of statistical model checking techniques.

Several statistical model checkers have been built and reported. As the focus of this paper lies towards real-time stochastic models, only a few model checkers apply. The PLASMA tool [16,9] can analyze stochastic models such as Continuous Time Markov Chains (CTMCs) [4] and Markov Decision Processes (MDPs). It however does not support real-time aspects (yet), so it is omitted in the remainder of this paper. UPPAAL-SMC [11] and Modes [7] do support the aforementioned real-time stochastic formalisms and are therefore within the scope of this paper.

3 Semantic Caveats

As statistical model checking builds upon Monte-Carlo simulations, a fully probabilistic interpretation is required of the real-time stochastic model. This involves inferring or assuming probabilistic distributions on model elements that have no probabilistic semantics. The implications of this are however not always made obvious to the user. We investigated this and summarized the results as a set of *semantic caveats* which are elaborated upon in this section.

These caveats can be distinguished into two groups: The first group contains the caveats that originate from resolving underspecifications in the model, which are caveats $\mathbf{C_1}$, $\mathbf{C_2}$ and $\mathbf{C_3}$. The second group contains caveats $\mathbf{C_4}$ and $\mathbf{C_5}$, which the caveats that arise due to possible inconsistencies in the semantics of the model after inferring or assuming the probability distributions.

The diagrams in this section are denoted as follows: Nodes correspond to locations in the model, where the upper part indicates the label of the node; the lower part indicates either the invariant associated with the location, or the exit rate of the location. Edges between nodes represent discrete transitions. The enabled time interval induced by the guard of a transition is shown below the edge, the action above. Each example has an associated global clock. In order to simplify the diagrams, this clock has been hidden from the invariants and guards.

$\mathbf{C_1}$ – Underspecified Scheduling between Processes

The range of possible interleavings between concurrent processes is typically left underspecified during the design of the system and is subject to refinement in subsequent development phases. As such, we are dealing with an underspecification of the possible execution schedule(s). This ensures a conservative over-approximation of the possible behaviors in the analysis results.

Statistical model checkers employ a probabilistic interpretation over this form of underspecification. However, it is not immediately clear what this interpretation should be based on. Depending on how the one-step probabilities are calculated, this "progress" could depend on a process with the shortest waiting time (possibly uniformly chosen among multiple waiting times), but may also be distributed over any process that is capable of making progress after waiting. Finally, samples from the waiting time distribution may be discarded entirely if a process fails to find an enabled transition, requiring the sample process to start over.

$\mathbf{C_2}$ – Underspecified Choice within a Process

Within the context of communicating concurrent processes, we identify two kinds of choice, namely *internal* and *external* choice. Internal choice is when multiple internal transitions emanate from a location in the same process and they are unobservable by other processes, yet they lead to different successor locations. With external choice, we have multiple synchronizing transitions emanating from a single location.

When either choice is underspecified within a single process, i.e. $Distr(T)$ is not specified, the statistical model checker has to provide a probabilistic interpretation over it (see e.g. [13]). This is typically implicitly interpreted as equiprobabilistic, but other approaches exist, see e.g. [18] and Section 5. Issues may arise due to the introduction of a bias towards certain behavior.

Consider the example in Figure 1, with an external choice between actions α and β, and an internal choice between two α transitions. Applying a strictly uniform distribution over all transitions introduces a bias towards action α. However, applying a uniform distributions over the actions first introduces a bias towards the transition towards location p_3.

$\mathbf{C_3}$ – Underspecified Waiting Times

Transition guards and invariants are powerful modeling concepts for expressing timed behavior. They express the range of time in which certain behavior can occur, without requiring the user to specify exact time points or delay distributions. However, in the case of such underspecifications, the statistical model checker has to provide one as well, similar to caveat $\mathbf{C_2}$. As an example, Figure 2 shows a process with a transition that is enabled in the interval $[1, 5]$, and an invariant that is true for all clock values in the interval $[0, 5]$, without specifying a delay distribution.

Delay distributions may be derived from any combination of invariants, transition guards and process synchronization. Generally, a uniform distribution is derived (like in [3]), though other distributions may be used or even required, see caveat $\mathbf{C_4}$. Again, care has to be taken not to introduce any unwanted bias. Choosing whether or not to derive a distribution based on transition guards or process communication influences this, see Section 4.

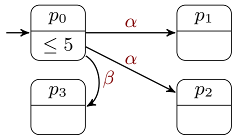


Fig. 1. Combination of internal and external choice

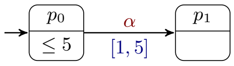


Fig. 2. Transition enabled within an interval

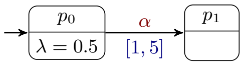


Fig. 3. Use of an exponential distribution

C_4 – Choice of Distributions

The previous caveats discussed underspecifications for which a statistical model checker has to generate probability distribution functions. However, based on the structure of the model, different distributions may be applicable. Often uniform distributions (both discrete and continuous) are employed, based on the notion of equiprobability. However, other distributions may be used or even required, for instance if the model specifies a boundary that extends to infinity, or specifies a deterministic delay by means of a point interval.

This introduces a caveat that requires attention in two directions. First, the support may not coincide with the enabledness of all transitions, which may lead for example to a deadlock (see also the accuracy dimension in Section 5). For instance, Figure 3 exemplifies this with an exponential distribution of which the support does not match the interval for which the transition is enabled. Second, and more obvious, the choice of distribution directly influences the bias on the waiting time.

C_5 – Invalid Paths

The modeling formalisms we consider come with an innate degree of abstraction. This comes with caveats by itself, namely Zeno behaviors, action-locks and deadlocks, which may generate semantically invalid paths. These caveats are well-known and widely studied [20]. Akin to analytical model checking techniques, also for statistical model checking these caveats require serious consideration.

Zeno behaviors may occur as a result of the model not allowing time to progress beyond a certain point, a timelock, or allowing paths that execute an infinite number of actions in a finite amount of time. As a result, a statistical model checker may not terminate its path generation when the termination condition is based on reaching a certain time bound.

Action-locks occur in the discrete part of the model, when the current state does not allow any further discrete transitions to be taken for any given delay.

Deadlocks are the combination of action-locks and timelocks. In a deadlocked state, neither the timed part nor the discrete part of the model can progress, and path generation will terminate prematurely. Examples of all three behaviors are shown in Figure 4. Note that for the action- and deadlocks, the behavior occurs in locations q_1 and r_1 respectively.

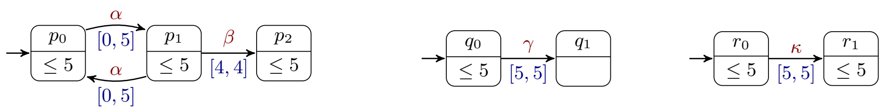


Fig. 4. Example processes showing respectively Zeno behavior, an action-lock and a deadlock

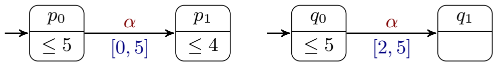


Fig. 5. Example network with synchronizing actions

4 Caveat Interactions

Thus far the caveats from the previous section have only been treated as separate instances. However, it is very well possible that they appear in the model at the same time, possibly constraining the possible distributions, or affecting the overall bias of the outcomes. Therefore, in this section we provide further examples to highlight the fact that caveats may interact and should not be considered as isolated entities.

$\mathbf{C_1}$,$\mathbf{C_3}$ – Event Synchronization with Partially Overlapping Enabledness Intervals

Underspecification of scheduling is generally dealt with by scheduling the process with the shortest waiting time first. Action synchronization requires two processes to take a discrete transition at the same time. Figure 5 highlights such a synchronization. Two processes P (left) and Q (right) are defined. Here, the action α is part of the communication alphabet. Thus the transitions synchronize and are therefore only enabled in the interval $[2, 4]$, due to the interaction of the guards and the invariants of the target locations.

This example highlights that action synchronization requires the path generator to consider all processes with synchronizing actions in order to prevent generating invalid paths that may for example invalidate invariants. For example, if only the local transitions in process P is considered, a waiting time of one may be sampled. However, it cannot synchronize then with process Q, as the transition is only enabled after time point two.

$\mathbf{C_2}$,$\mathbf{C_3}$ – Bias of Time Intervals

The bias towards a certain transition greatly depends on the time intervals. Figure 6 shows three different types of intervals that are subject to underspecification of choice as well. In all cases, if the underspecification of choice is resolved first and uniformly, selecting each outgoing transition with probability $\frac{1}{2}$, the probability of reaching either of the two target states is equal.

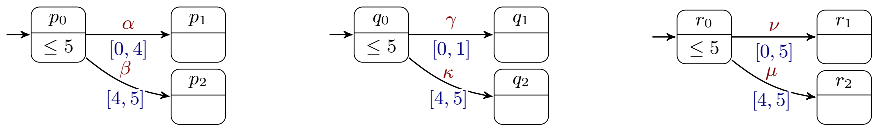


Fig. 6. Examples of underspecification of both choice and time with convex, non-convex and overlapping intervals respectively

However, resolving the underspecification of time first can cause considerable differences. In the first example (Figure 6, left), sampling in the interval in which either transition is enabled will cause a bias towards transition α (with probability $\frac{4}{5}$), as it has a larger interval. In the second case, both intervals are equally large. However, due to the union of the intervals being non-convex, a bias may be formed towards transition κ if the simulator generates a delay in the interval $[1, 4]$ as transition κ is the only transition that will become enabled after such a time point (see also Section 5.2). The third case shows the effect of the *scope* of the samples (Section 5.3). If a single sample is generated for location r_0, the waiting time is uniformly distributed. On the other hand, when generating a sample for both transitions individually, the probability distribution over the time intervals is no longer uniform, as the interval $[4, 5]$ of transition μ will increase the likelihood of picking a delay from that interval (when resolving the choice first (Section 5.1), the probability of a delay in $[4, 5]$ would become $\frac{1}{2} \cdot \frac{1}{5} + \frac{1}{2} \cdot \frac{1}{1} = \frac{3}{5}$).

5 Classifying SMC Algorithms

All statistical model checkers encounter the caveats outlined in the previous two sections and their implementations somehow deal with them. There is a range of possible solutions. We analyzed them and developed a classification scheme that has four dimensions, which we elaborate upon in the next sections.

5.1 Transition Selection Order – Early versus Delayed

The order solution dimension follows from caveats $\mathbf{C_1}$ and $\mathbf{C_2}$. It relates to the moment an enabled transition is chosen with respect to the moment that the waiting times are sampled. The choice of a particular order also impacts the accuracy dimension, which is elaborated later on.

Early ordering picks from each process a single enabled transition before sampling a waiting time. Thus, the race between transitions within a process is not determined by the execution times of the individual transitions. In our setting, this decision is usually made equiprobabilistically.

In the case of *delayed* ordering, the *waiting time(s)* are sampled first, simulating racing transitions. After sampling, the algorithm picks a winning transition which is used to extend the path. This choice is typically made equiprobabilistically between the fastest transitions, similar to racing processes.

Taking Figure 1 as an example, early ordering would pick either of the 3 transitions – usually uniformly – and sample a delay afterward. Delayed ordering does the opposite and generates a delay based on location p_0 and then pick any of the (enabled) transitions, possibly based on the delay.

A hybrid approach exists as well. In the work of [18], the preselection policy allows a set of transitions to be preselected, probabilistically or deterministically, which may then enter the race between processes. The simulation process then continues further as with delayed ordering.

5.2 Accuracy of Waiting Time Distributions – Exact versus Approximate

This dimension follows from caveat $\mathbf{C_3}$, the underspecification of time. This underspecification is resolved using a probability distribution over the possible time intervals. The simulator may generate such a distribution with a support that exactly coincides with the time intervals in which any transition is enabled, or it may approximate these intervals. Information used may stem from source and target location invariants, and transition guards. Synchronization effects may be taken into account as well.

In the *exact* case, the constructed probability distribution has a support that matches the time points that enable at least one transition. This is a strong property. Note that the constructed distribution can be non-convex. In the example of Figure 6 in the middle, this would mean the sampled waiting time is generated in the set $[0, 1] \cup [4, 5]$.

The *approximate* case allows the distribution to range over an over- or under-approximation of the set of possible waiting times. That means waiting times can be sampled for which no transition is enabled. Taking again the middle process of Figure 6 as an example, based only on the location invariant a possible interval could be $[0, 5]$. Over-approximating potentially results in an action-lock (see caveat $\mathbf{C_5}$). It is then up to the SMC algorithm what to do with the current sampled waiting time. This is the attitude policy and is discussed in Section 5.4.

5.3 Scope of Waiting Time Samples – Location Local versus Transition Local

The scope dimension follows from caveats $\mathbf{C_2}$ and $\mathbf{C_3}$ and when multiple transitions are enabled. A single delay can be sampled for the current location, or individual delays can be sampled for each transition emanating from the current location. Such a situation exists in all examples of Figure 6, where one delay may be generated entailing both transitions, or one delay per transition, totaling two.

We refer to the former solution as the *location local* scope. The distribution that is being constructed will account for all enabled time intervals of all non-deterministic choices. A waiting time sampled from that distribution does not necessarily imply which transition progresses, as the non-deterministic choices may have overlapping enabled intervals. As only one sample is generated for all transitions, the location local scope introduces a bias towards transitions that

Table 1. Summary of algorithmic policies

Policy class	Transition	Policies keeping sample	Policies rejecting sample
Memory	Enabled	Age & Enabling Memory	Resampling
	Disabled	Age Memory	Enabling Memory
Attitude	Enabled	Progressive & Conservative	-
	Disabled	Progressive	Conservative

have a larger interval, and may in fact render the simulator incapable of picking transitions with a deterministic point interval.

The alternative is referred to as the *transition local* scope. Here a sample is generated for each non-deterministic choice from a process' current location. Then, a race occurs within the process between the individual transitions. The transition local scope may introduce a bias as well. As was shown in Section 4 (Figure 6 right example), different transition intervals may induce a bias within the overall distribution of delays, in this case towards $[4, 5]$.

5.4 Race Policy

The race policy dimension addresses the lifetime of sample(s) generated for a process under a race condition. When the stochastic model is described by non-memoryless distributions, different probabilistic outcomes can be realized based on the manner in which samples are retained or discarded, which is decided by a policy. Such a policy can make different decisions based on winning or losing the race described by caveat $\mathbf{C_1}$ and whether any associated transitions are enabled or not.

Memory policies describe whether or not the generated sample will be retained for a future step if the process *lost* the race. In [18] three policies are described: age memory, which retains the sample if there is no enabled transition; enabling memory, which rejects the sample if no transition is enabled; and resampling, which always rejects the sample. Furthermore, the age and enabling policies keep the sample if the transition remains enabled.

Attitude policies dictate what samples are valid for a process that *won* the race. The conservative policy discards the sample if no transition is enabled. The progressive policy keeps the sample even if no transition is enabled (potentially causing an action-lock, see caveat $\mathbf{C_5}$). Both policies keep the sample (and use it) when a transition is enabled, and no policy rejects a sample in such a case. Both the memory and attitude policies are summarized in Table 1.

6 Applying the Systematization

To put the systematization in this paper to practical terms, we show how UPPAAL-SMC and MODES deal with the caveats and how their respective

Table 2. Model checking results from UPPAAL-SMC and MODES on the examples. The UPPAAL-SMC and MODES columns show the calculated probability intervals.

Caveats	Example	Property	UPPAAL	MODES
$\mathbf{C_2}$	Figure 1	$\mathbb{P}(\diamond^{[0,5]}(p_1 \vee p_2))$	[0.66,0.68]	[0.66,0.68]
		$\mathbb{P}(\diamond^{[0,5]}p_3)$	[0.32,0.34]	[0.32,0.34]
$\mathbf{C_3}$	Figure 2	$\mathbb{P}(\diamond^{[0,4]}p_1)$[a]	[0.74,0.76]	[0.98,1.00]
$\mathbf{C_4}$	Figure 3	$\mathbb{P}(\diamond^{[0,5]}p_1)$	[0.86,0.88]	[0.51,0.53]
$\mathbf{C_1}$,$\mathbf{C_3}$	Figure 5	$\mathbb{P}(\diamond^{[0,5]}(p_1 \wedge q_1))$	error[b]	[0.98,1.00]
$\mathbf{C_2}$,$\mathbf{C_3}$	Figure 6	$\mathbb{P}(\diamond^{[0,5]}p_1)$	[0.79,0.81]	[0.98,1.00]
		$\mathbb{P}(\diamond^{[0,5]}p_2)$	[0.19,0.21]	[0.00,0.02]
$\mathbf{C_2}$,$\mathbf{C_3}$ Non-convex	Figure 6	$\mathbb{P}(\diamond^{[0,5]}q_1)$	[0.19,0.21]	[0.98,1.00]
		$\mathbb{P}(\diamond^{[0,5]}q_2)$	[0.79,0.81]	[0.00,0.02]
$\mathbf{C_2}$,$\mathbf{C_3}$ Overlapping	Figure 6	$\mathbb{P}(\diamond^{[0,5]}r_1)$	[0.89,0.91]	[0.98,1.00]
		$\mathbb{P}(\diamond^{[0,5]}r_2)$	[0.09,0.11]	[0.00,0.02]

[a] A time bound of four was chosen to highlight the difference between UPPAAL-SMC and MODES scheduling.
[b] UPPAAL-SMC cannot execute the model as it is not input enabled.

SMC implementations can be classified. We furthermore compared these two tools using the examples presented in Sections 3 and 4. These examples have been modeled using the formalism used by the respective tool, and probabilistic time bounded reachability properties were evaluated to quantify the differences. The results are presented in Table 2.

For both tools, results were determined with a 0.99 confidence, and a 0.01 error bound. These parameters were chosen as they provide sufficient precision and confidence to compare the outcomes of the experiments. Note that due to the existence of an error bound, the results are not exact values but rather intervals. More detailed results and the sources of the models can be found online [10].

UPPAAL-SMC vs MODES

The path generation algorithm implemented by UPPAAL-SMC is described in [11]. The algorithm first determines the interval of possible waiting times by inspecting the invariant of the current active location. It then delays based on a sample from that interval, and uniformly chooses from the enabled transitions afterward. Any previously generated samples are ignored. When multiple processes are involved, the sample with shortest waiting time is selected.

The behavior of the path generation algorithm in the MODES tool [7] has been analyzed using the semantics of the MODEST language, configuration of the MODES tool and analysis of the experimental results. No information has been provided by the authors or derivative work on the path generation algorithm of

Modes. The Modes tool allows the resolution for both the underspecification of choice and time to be configured. Underspecification of choice can be resolved in four ways: The model is rejected; confluence detection is used to remove spurious non-determinism [14]; partial order reduction is used to remove spurious non-determinism [6]; or a uniform distribution is applied. Underspecification of waiting time can be resolved in two ways: Either the model is rejected, or an as-soon-as-possible (ASAP) scheduler is used (which always selects the shortest possible waiting time to enable a transition). For the experiments performed in this section, the uniform distribution is used for underspecification of choice, and the ASAP scheduler for underspecification of time.

Discussion

Most of the differences in the results can be attributed to the ASAP scheduler of the Modes tool. Despite having a delayed and approximate scheduling algorithm, the Modes tool always makes a deterministic choice to sample the shortest possible waiting time. Thus, whereas the results from UPPAAL-SMC tend towards a uniform distribution when choice is involved, Modes tends towards a Dirac-delta distribution. This can be seen for all the cases involving $\mathbf{C_3}$ (Figures 2, 5 and 6). Here, the probability of reaching a certain state within the specified time bound is approximately either 1.0 or 0.0 for Modes, indicating the chosen delay is constant.

Both the UPPAAL-SMC and Modes tools make use of a delayed order, resolving the underspecification of time before selecting a transition. For Modes, this can be seen in the results of the $\mathbf{C_3}$ case, as only the transition with the lowest time bound is chosen (due to the ASAP scheduler). In this case, Modes always chooses a delay of zero and thus, considering the examples in Figure 6, ends up in locations p_1, q_1 and r_1 respectively. Furthermore, both tools have an approximate accuracy. UPPAAL-SMC only takes the invariants into account, over-approximating the possible transition times. Modes only uses the lowest possible time value, thus under-approximating the possible transition times. For the scope dimension, both tools use the location local scope: UPPAAL-SMC uses just the invariants of a location to generate this sample whereas Modes picks the lowest possible value.

Both tools differ in the applied race policies. In case of UPPAAL-SMC, a resampling memory policy is used, as previous results are discarded when a new step is generated, and a progressive attitude policy is used, as a generated delay is always applied even if no transition is enabled. In the case of Modes, when process loses the race the age memory policy applies, as samples are explicitly assigned to process variables. However, the attitude policy can not be determined, as the scheduler ensures that there is always an enabled transition, as the difference can only be determined for delays after which no transition is enabled.

An interesting difference in results can be seen for the $\mathbf{C_4}$ case (Figure 3). This can be attributed to a difference in semantics for models containing exponential distributions to sample waiting times. In the example, location p_1 can only be reached in the interval $[1, 5]$. Both Modes and UPPAAL-SMC interpret any sample above five as a deadlock. However, there is a difference in the interpretation of values below one. In UPPAAL-SMC, the outcome of the entire exponential distribution is shifted to the right by one unit of time, such that a waiting time below one is never generated. In Modes, a waiting time below one simply results in a deadlock, explaining the lower probability of reaching p_1.

7 Conclusions

The light-weight and scalable nature of statistical model checking techniques appeal as a practical way to handle the rich semantics of real-time stochastic models. From experimentation with two publicly available SMC tools, UPPAAL-SMC and Modes, we however encountered discrepancies in the computed probabilities on structurally equivalent models. They could not easily be dismissed as tool implementation bugs. In fact, they turned out to be semantic biases that did not align with our end-user interpretation. In our effort to study and understand these biases, we investigated how statistical model checkers deal with concurrency, non-deterministic and real-time aspects and how the discrepancies in the probabilities can be traced to key SMC tool design and implementation choices. From the lessons learned in this investigation, we systematized our observations into five caveats and a classification scheme for SMC algorithms. They can be used to understand any SMC technique on real-time stochastic models. We furthermore exemplify the caveats with concrete models, and show and discuss how two publicly available SMC tools, UPPAAL-SMC and Modes, compute significantly different probabilities on them.

We restrict our conclusion to the following: the use of SMC techniques on real-time stochastic models needs to be approached with caution. The systematization in this paper helps the end-user to identify and deal with the key points for caution. In the end, this increased understanding of SMC techniques helps the end-user to interpret the (difference in) computed probabilities by SMC tools. Their perceived biases ought not to be straightforwardly seen as an invalidation of SMC techniques. In fact, these biases may for example perfectly suit the assumptions that hold in the modeling domain (e.g. RAMS or systems biology). This paper contributes with a systematization for validating the SMC tools' behaviors against these desired assumptions, which might be for example used for tool certification.

References

1. Alur, R., Dill, D.L.: A Theory of Timed Automata. Theoretical Computer Science 126(2), 183–235 (1994)
2. Alur, R., La Torre, S., Pappas, G.J.: Optimal Paths in Weighted Timed Automata. In: Di Benedetto, M.D., Sangiovanni-Vincentelli, A.L. (eds.) HSCC 2001. LNCS, vol. 2034, pp. 49–62. Springer, Heidelberg (2001)

3. Baier, C., Bertrand, N., Bouyer, P., Brihaye, T., Größer, M.: Probabilistic and Topological Semantics for Timed Automata. In: Arvind, V., Prasad, S. (eds.) FSTTCS 2007. LNCS, vol. 4855, pp. 179–191. Springer, Heidelberg (2007)
4. Baier, C., Haverkort, B., Hermanns, H., Katoen, J.-P.: Model Checking Algorithms for Continuous-Time Markov Chains. IEEE Transactions on Software Engineering 29(6), 524–541 (2003)
5. Behrmann, G., Fehnker, A., Hune, T., Larsen, K.G., Pettersson, P., Romijn, J., Vaandrager, F.W.: Minimum-Cost Reachability for Priced Timed Automata. In: Di Benedetto, M.D., Sangiovanni-Vincentelli, A.L. (eds.) HSCC 2001. LNCS, vol. 2034, pp. 147–161. Springer, Heidelberg (2001)
6. Bogdoll, J., Ferrer Fioriti, L.M., Hartmanns, A., Hermanns, H.: Partial Order Methods for Statistical Model Checking and Simulation. In: Bruni, R., Dingel, J. (eds.) FMOODS/FORTE 2011. LNCS, vol. 6722, pp. 59–74. Springer, Heidelberg (2011)
7. Bogdoll, J., Hartmanns, A., Hermanns, H.: Simulation and Statistical Model Checking for Modestly Nondeterministic Models. In: Schmitt, J.B. (ed.) MMB & DFT 2012. LNCS, vol. 7201, pp. 249–252. Springer, Heidelberg (2012)
8. Bohnenkamp, H., D'Argenio, P.R., Hermanns, H., Katoen, J.-P.: MODEST: A Compositional Modeling Formalism for Hard and Softly Timed Systems. IEEE Transactions on Software Engineering 32(10), 812–830 (2006)
9. Boyer, B., Corre, K., Legay, A., Sedwards, S.: PLASMA-lab: A Flexible, Distributable Statistical Model Checking Library. In: Joshi, K., Siegle, M., Stoelinga, M., D'Argenio, P.R. (eds.) QEST 2013. LNCS, vol. 8054, pp. 160–164. Springer, Heidelberg (2013)
10. Bruintjes, H., Nguyen, V.Y.: Test results from experiments, `http://www-i2.informatik.rwth-aachen.de/~isola2014/smc/index.html` (Online; accessed May 14, 2014)
11. Bulychev, P., David, A., Larsen, K.G., Mikučionis, M., Bøgsted Poulsen, D., Legay, A., Wang, Z.: UPPAAL-SMC: Statistical Model Checking for Priced Timed Automata. In: Wiklicky, H., Massink, M. (eds.) QAPL. Electronic Proceedings in Theoretical Computer Science, vol. 85, pp. 1–16. Open Publishing Association (2012)
12. Fränzle, M., Hahn, E.M., Hermanns, H., Wolovick, N., Zhang, L.: Measurability and Safety Verification for Stochastic Hybrid Systems. In: HSCC 2011, pp. 43–52. ACM (2011)
13. Grosu, R., Smolka, S.A.: Monte Carlo Model Checking. In: Halbwachs, N., Zuck, L.D. (eds.) TACAS 2005. LNCS, vol. 3440, pp. 271–286. Springer, Heidelberg (2005)
14. Hartmanns, A., Timmer, M.: On-the-Fly Confluence Detection for Statistical Model Checking. In: Brat, G., Rungta, N., Venet, A. (eds.) NFM 2013. LNCS, vol. 7871, pp. 337–351. Springer, Heidelberg (2013)
15. Hermanns, H. (ed.): Interactive Markov Chains. LNCS, vol. 2428. Springer, Heidelberg (2002)
16. Jegourel, C., Legay, A., Sedwards, S.: A Platform for High Performance Statistical Model Checking – PLASMA. In: Flanagan, C., König, B. (eds.) TACAS 2012. LNCS, vol. 7214, pp. 498–503. Springer, Heidelberg (2012)

17. Legay, A., Delahaye, B., Bensalem, S.: Statistical Model Checking: An Overview. In: Barringer, H., Falcone, Y., Finkbeiner, B., Havelund, K., Lee, I., Pace, G., Roşu, G., Sokolsky, O., Tillmann, N. (eds.) RV 2010. LNCS, vol. 6418, pp. 122–135. Springer, Heidelberg (2010)
18. Marsan, M.A., Balbo, G., Bobbio, A., Chiola, G., Conte, G., Cumani, A.: The Effect of Execution Policies on the Semantics and Analysis of Stochastic Petri Nets. IEEE Transactions on Software Engineering 15(7), 832–846 (1989)
19. Sproston, J.: Decidable Model Checking of Probabilistic Hybrid Automata. In: Joseph, M. (ed.) FTRTFT 2000. LNCS, vol. 1926, pp. 31–45. Springer, Heidelberg (2000)
20. Tripakis, S.: Verifying Progress in Timed Systems. In: Katoen, J.-P. (ed.) ARTS 1999. LNCS, vol. 1601, pp. 299–314. Springer, Heidelberg (1999)
21. Younes, H.L., Simmons, R.G.: Statistical Probabilistic Model Checking With a Focus on Time-Bounded Properties. Information and Computation 204(9), 1368–1409 (2006)

Formal Analysis of the Wnt/β-catenin Pathway through Statistical Model Checking

Paolo Ballarini[1], Emmanuelle Gallet[1],
Pascale Le Gall[1], and Matthieu Manceny[2]

[1] Laboratoire MAS, Ecole Centrale Paris, 92195 Châtenay-Malabry, France
{emmanuelle.gallet,pascale.legall,paolo.ballarini}@ecp.fr
[2] Laboratoire LISITE, ISEP, 28 Rue Notre-Dame-des-Champs 75006 Paris, France
matthieu.manceny@isep.fr

Abstract. The Wnt/β-catenin signalling pathway plays an important role in the proliferation of neural cells, and hence it is the main focus of several research aimed at understanding neurodegenerative pathologies. In this paper we consider a compact model of the basic mechanisms of the Wnt/β-catenin pathway and we analyse its dynamics by application of an expressive temporal logic formalism, namely the Hybrid Automata Stochastic Logic. This allows us to formally characterise, and effectively assess, sophisticated aspects of the Wnt/β-catenin pathway dynamics.

Keywords: HASL Model Checking, Stochastic modelling, biological pathways, Wnt/β-catenin.

1 Introduction

Systems Biology [11] is concerned with the development of formalisms for building "realistic" models of biological systems, i.e. models capable of reproducing wet-lab observations. A biological model consists of a set biochemical agents (i.e. species) whose interactions are expressed by a set of *reaction equations.* This leads to either a continuous-deterministic interpretation (i.e. in terms of a system of differential equations), or to a discrete-stochastic interpretation (i.e. in terms of a discrete-state stochastic process).

Stochastic modelling and systems biology. Within the discrete-stochastic semantics realm, which is what we consider in this work, molecular interactions are assumed to be of stochastic nature hence biochemical reactions occur according to probability distributions. In this case what modellers normally do is to generate one (or several) trajectory(ies) through stochastic simulation and observe the evolution of the species (under different model's configurations) in order to figure out how a given aspect of the model's dynamics is affected by the various elements of the model (i.e. what species/reactions is responsible for a given observed behaviour). Such an approach has two main advantages: its simplicity and its low computational cost (the runtime for generating a single trajectory or a normally small number of trajectories is very low even for large models). On the other hand the main disadvantage is that it is little formal, meaning that the modeller must draw conclusions based only on the observation of a single (stochastic) trajectory (or of a trajectory obtained by averaging a normally small number of trajectories).

Stochastic model checking and systems biology. Stochastic model checking [12] (SMC) is a formal technique that allows the modeller to formally express relevant properties in terms of a (stochastic) temporal logic and to assess them against a given stochastic model. This is achieved through an automatic procedure which can either provide an *exact answer* through exhaustive exploration of the model's state space (i.e. *numerical model checking* [2]) or an *estimated answer* resulting from a finite sampling of the model's trajectory (i.e. *statistical model checking* [13]). SMC has at least two main advantages with respect to informal approaches: first it provides the modeller with a language for capturing relevant properties formally; second the answer it calculates (e.g. probability that a property is satisfied by the model) are either exact (i.e. they reflect the complete set of possible behaviours of the model) or are accurate estimates (i.e. calculated over a a sufficiently large sample of trajectories). The effectiveness of SMC in systems biology applications is demonstrated by an ever increasing number of publications, e.g.[8,10,5].

β-catenin and the WNT Pathway. In cellular biology signalling pathways are basic mechanisms responsible for controlling a cell's life-cycle. Simply speaking a signalling pathway represents a cascade of biochemical reactions which is triggered by a specific signal (i.e. type of molecules) whose presence, normally at the cell membrane, activates the cascade leading to the "transmission" of the signal inside the cell (i.e. cytosol and/or nucleus). In this paper we study a model of the Wnt/β-catenin pathway, a signalling pathway known to be involved in the pathological degeneration of neuronal cells [14].

Our Contribution. In this work we present preliminary results of application of formal analysis, based on the so-called Hybrid Automata Stochastic Logic (HASL) statistical model checking, to a model of the Wnt/β-catenin pathway presented in [15]. In particular we show how one can define specific HASL formulae for assessing sophisticated characteristics of the Wnt/β-catenin pathway dynamics. This includes, for example, measuring the temporal location and the amplitude of transient peaks of nuclear β-catenin, exhibited by certain initial conditions, or assessing its oscillatory character resulting from other conditions. If in [15] the analysis of the Wnt/β-catenin model is simply done through plotting of simulated trajectories, here we move analysis to a higher and more formal level by demonstrating how, through model checking, one gains access to the analysis of sophisticated dynamical aspects of the Wnt/β-catenin pathway.

Paper Organisation. We introduce the Wnt/β-catenin mechanism in Section 2 and describe the model presented in [15] which we have used for our analysis. In

Section 3 we give a concise description of the HASL statistical model checking formalism. In Section 4 we present the results obtained by application of HASL model checking to the analysis of Wnt/β-catenin model. We wrap up the paper with some conclusive remarks and future perspectives in Section 5.

2 A Model of the Wnt/β-catenin Pathway

Neurodegeneration is the process of progressive lost of structure/function of neuronal cells (i.e. neurons) which is at the basis of many neurodegenerative diseases, such as, for example, the Parkison's disease, Alzheimer's disease and the Amyotrophic lateral sclerosis. Research in this field is particularly focused on the growth of *in vitro* population of neural cells that may potentially be used in replacement therapies for neurodegenerative diseases. Cultivated cells undergo so-called proliferation, a process of successive cell divisions and potential differentiation into neurones and glial cells.

The Wnt/β-catenin pathway is a signalling pathway known to be involved in the proliferation/differentiation of neural cells. Specific in vitro experiments [14] have exhibited a high activity of the Wnt/β-catenin pathway during the differentiation of ReNcell VM (RVM) cells, i.e. a type of cells derived from the brain of a fetus and that are believed to be an appropriate model for replacement therapies in neurodegenerative pathologies. The activity of the Wnt/β-catenin is summarised as follows: in absence of extracellular Wnt molecules (normally at cell's membrane), a degradation complex causes the phosphorylation and subsequent destruction of β-catenin located in the cell's cytosol (denoted β_{cyt}); on the other hand in presence of Wnt proteins, the degradation complex is inactivated resulting in accumulation of β_{cyt}. Furthermore from the cytosol β-catenin undergoes a (reversible) relocation to the nucleus (denoted β_{nuc}) wherein it activates the expression of one component of its degradation complex, i.e. the Axin protein. The above described mechanism is captured by a core-version of the Wnt/β-catenin pathway model presented in [15]. This consists of the twelve biochemical reactions illustrated by equations (1).

$$
\begin{aligned}
&R_1 : Wnt \xrightarrow{k_1} \varnothing && R_7 : AxinP + \beta cyt \xrightarrow{k_7} AxinP\\
&R_2 : Wnt + AxinP \xrightarrow{k_2} Wnt + Axin && R_8 : \varnothing \xrightarrow{k_8} \beta cyt\\
&R_3 : AxinP \xrightarrow{k_3} Axin && R_9 : \beta cyt \xrightarrow{k_9} \varnothing\\
&R_4 : Axin \xrightarrow{k_4} AxinP && R_{10} : \beta cyt \xrightarrow{k_{10}} \beta nuc\\
&R_5 : AxinP \xrightarrow{k_5} \varnothing && R_{11} : \beta nuc \xrightarrow{k_{11}} \beta cyt\\
&R_6 : Axin \xrightarrow{k_6} \varnothing && R_{12} : \beta nuc \xrightarrow{k_{12}} Axin + \beta nuc
\end{aligned}
\tag{1}
$$

The model consists of three basic molecular species: the Axin protein, which can be either in normal ($Axin$) or phosphorylated ($AxinP$) form, the Wnt protein (Wnt) and the β-catenin which can be either located in the cytosol

(β_{cyt}) or in the nucleus (β_{nuc}). Equations (1) account for the following aspects: two reversible events, i.e. the phosphorylation of Axin (reactions R_4 and R_3) and the relocation of β-catenin from/to cytosol/nucleus (reactions R_{10} and R_{11}); the Wnt enhanced de-phosphorylation of $Axin$ (reaction R_2); the nuclear β-catenin (i.e. β_{nuc}) regulated expression of $Axin$ (reaction R_{12}); the phosphorylated $AxinP$ enhanced degradation of cytosolic β-catenin (i.e. β_{cyt}) (reaction R_7)[1]; the constant (DNA regulated) expression of cytosolic β-catenin (i.e. β_{cyt}) (reactions R_8); the degradation of all species i.e. Wnt (reactions R_1), Axin in either form (reactions R_5 or R_6) and β_{cyt} (reaction R_9).

In this paper we focus on the discrete-stochastic interpretation of Equations (1), hence species populations are expressed in terms of number of molecules and reactions are of stochastic nature and are assumed to obey the *mass action* law (meaning that a reaction's rate is proportional to the current population of the reactants, except for R_8 whose rate is constant). With respect to the model configuration we consider two basic sets A and B of parameter values respectively taken from the sets 3 and 4 in [15], and indicated in Table 1.

Table 1. Parameter sets for stochastic interpretation of Wnt/β-catenin pathway model given by equations (1)

initial populations ($mol.$)			rate constants ($mol. \cdot min^{-1}$)					
par. id	**Set A**	**Set B**	par. id	**Set A**	**Set B**	par. id	**Set A**	**Set B**
nβcyt	11145	12989	k_1	0.6	0.27	k_6	$2.4 \cdot 10^{-3}$	$4.48 \cdot 10^{-3}$
nβ_{nuc}	4532	5282	k_2	10	20	k_7	$3 \cdot 10^{-4}$	$2.1 \cdot 10^{-4}$
nAxin	144	252	k_3	0.03	0.03	k_8	420	600
nAxinP	125	219	k_4	0.03	0.03	k_9	$1.13 \cdot 10^{-4}$	$1.13 \cdot 10^{-4}$
nWnt	1000	1000	k_5	$4.48 \cdot 10^{-3}$	$4.48 \cdot 10^{-3}$	k_{10}	0.0549	0.0549
			k_{11}	0.135	0.135	k_{12}	$2 \cdot 10^{-4}$	$4 \cdot 10^{-4}$

Throughout the remainder of the paper we will analyse the Wnt/β-catenin model by comparing the dynamics corresponding to the two parameter sets of Table 1. Following [15], we will also consider two variants of the *basic model* (1). The first variant, denoted *Wnt-inject*, represents a single injection of an extra amount (i.e. 1000) of Wnt molecules in the system at a fixed delay d_i. The second variant, denoted *Wnt-doped*, represents the presence of a doping mechanism that kicks in at a given delay d_d and then it sustainably generates a fresh Wnt molecule at given frequency (assumed to be exponential distributed with parameter k_d).

2.1 Stochastic Petri Net Model of the Wnt/β-catenin Pathway

The Cosmos [3] model checker which we used for analysing the Wnt/β-catenin pathway model uses Generalised Stochastic Petri Net (GSPN) [1] as modelling

[1] Notice that β_{nuc} dependent $Axin$ expression and $AxinP$ enhanced β_{cyt} degradation determine, *de facto*, a negative feedback loop between β-catenin and the Axin protein.

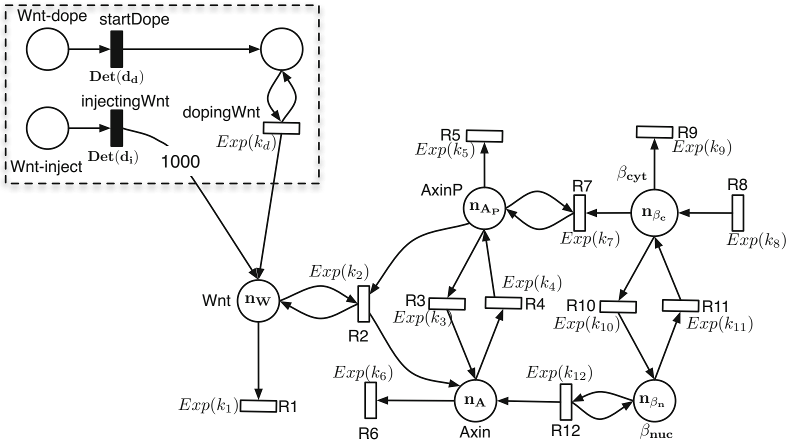


Fig. 1. GSPN model corresponding to Equations (1) of the Wnt/β-catenin pathway

formalism. A GSPN model is a bipartite graph consisting of two classes of nodes, *places* and *transitions*. Places (circle nodes) may contain *tokens* (representing the state of the modelled system) while transitions (bar nodes) indicate how tokens "flow" within the net (encoding the model dynamics). The state of a GSPN consists of a *marking* indicating the distribution of tokens throughout the places (i.e. how many tokens each place contains). A transition is enabled whenever all of its *input places* contains a number of tokens greater than or equal to the multiplicity of the corresponding (input) arc. An enabled transition may *fire* consuming tokens (in a number indicated by the multiplicity[2] of the corresponding input arcs) from all of its input places and producing tokens (in a number indicated by the multiplicity of the corresponding output arcs) in all of its output places. Transitions can be either *timed* (denoted by thick bars) or *immediate* (denoted by thin bars). Timed transitions are associated to a probability distribution (e.g. Exponential, Uniform, Deterministic, etc). In the context of this paper GSPN places represent biological species (and their marking the molecular population of a species), whereas timed transitions represent chemical reactions. For more details on GSPN we refer the reader to the literature [1].

Figure 1 depicts the GSPN model encoding the Wnt/β-catenin chemical equations (1). The net contains a place for each species of the Wnt/β-catenin model and a transition for each reaction. Non filled-in transitions are exponentially distributed (with marking dependent rate) with rate-constant corresponding to that of either parameter set of Table 1. The sub-net enclosed in dashed line box (top left) has been added in order to add the *Wnt-inject* and *Wnt-dope* behaviour to the basic model. In order to study the behaviour of the *Wnt-inject* (*Wnt-dope*) variant it suffices to add one token in the initial marking of place Wnt_inject

[2] The default multiplicity of an arc is 1 if different is explicitly indicated on the arc.

(*Wnt_dope*). Notice that the black filled-in transitions of the *Wnt-inject*/*Wnt-dope* subnet are associated to deterministic delays.

Observing β_{nuc} Dynamics on a Single Stochastic Trajectory. Following [15] we first look at the dynamics of nuclear β-catenin (i.e. β_{nuc}) as observed along a single trajectory simulated over 24 hours (Figure 2 and Figure 3 with the units on x axis being minutes). Figure 2 compares the behavior of β_{nuc} for the two parameter sets (Table 1) of the basic Wnt/β-catenin model in presence of Wnt (initial population of Wnt set to 1000).

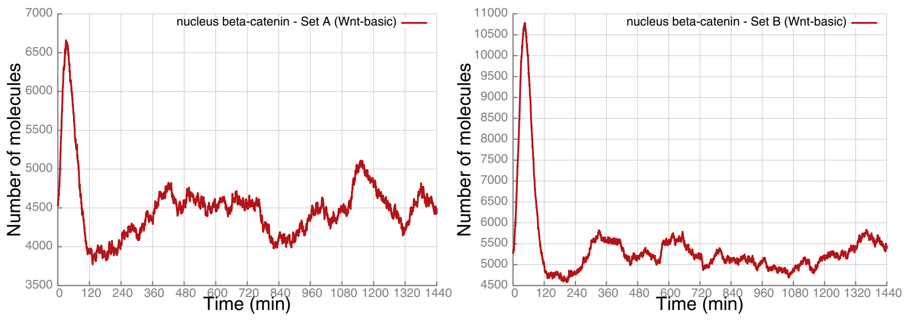


Fig. 2. Dynamics of β_{nuc} along a 24 hours single trajectory of the Wnt/β-catenin pathway model with parameter set A (left) and set B (right)

The interpretation in this case is quite straightforward: the presence of an initial Wnt signal (i.e. of 1000 molecules) triggers a (delayed) peak in β_{nuc} which however quickly ends due to the steady degradation (and absence of reintegration) of Wnt. Eventually when all Wnt has faded away β_{nuc} noisily converges to a certain level. Figure 3 compares the dynamics of β_{nuc} in presence

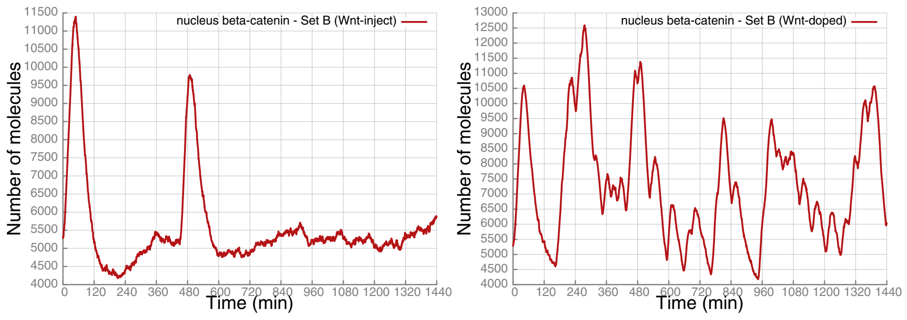


Fig. 3. Dynamics of β_{nuc} along a single trajectory of the Wnt/β-catenin pathway model with re-injection of Wnt at $t = 450$ minutes (left) and with doping of Wnt started at $t = 150$ minutes (right) and parameter set B

of *delayed injection* (i.e. *Wnt-inject* model injecting 1000 Wnt molecules at $d_i = 450$ minutes), and in presence of doping (i.e. modle *Wnt-dope* with doping starting at time $d_d = 150$ minutes). The effect of delayed injection of 1000 molecules of Wnt at time $d_i = 450$ minutes is highlighted by the presence of the second peak (Figure 3 left). On the other hand the effect of starting a persistent doping of Wnt at time $d_d = 150$ minutes results in an oscillatory, yet rather irregular, behaviour of β_{nuc} (Figure 3 right). In Section 4 we are going to illustrate how to take advantage of the HASL formalism for formally capturing the relevant dynamical characteristics of the above shown trajectories.

3 HASL Statistical Model Checking

The Hybrid Automata Stochastic Logic (HASL), introduced in [4], extends Deterministic Timed Automata (DTA) logics for addressing Markov chain models [9,7], by employing Linear Hybrid Automata (LHA) as instruments for addressing a general class of stochastic processes, namely that of Discrete Event Stochastic Processes (DESP). An HASL formula $\phi \equiv (\mathcal{A}, Z)$ consists of two elements: 1) $\mathcal{A}$, a *synchronising* LHA (i.e. an LHA enriched with DESP state and/or event *indicators*) and 2) Z a target expression (see grammar (2)) which expresses the quantity to be evaluated (either a measure of probability or, more generically, any real-valued measure).

Thus given a DESP model $\mathcal{D}$ and a formula $\phi \equiv (\mathcal{A}, Z)$ the HASL model checking procedure employs stochastic simulation to samples trajectories of the synchronised process $\mathcal{D}\times\mathcal{A}$, and then use (only) the paths *selected* by $\mathcal{A}$ (i.e. those paths of $\mathcal{D}\times\mathcal{A}$ that reach an accepting location of $\mathcal{A}$) for estimating the confidence-interval of the target measure Z. Such a procedure is implemented within the Cosmos [3] model checking framework, a tool which belongs to the fast expanding family of statistical model checkers (e.g. [6,16,17]). For practical reasons in HASL (and in particular within Cosmos) we employ GSPN as high-level language for expressing a DESP. Below we give the definition of DESP and LHA and informally describe the synchronisation process (we refer the reader to [4] for a more formal treatment).

Definition 1 (DESP). *A DESP is a tuple* $\mathcal{D} = \langle S, \pi_0, E, Ind, enabled, delay, choice, target\rangle$ *where*

- *S is a (possibly infinite) set of states,*
- *$\pi_0 \in dist(S)$ is the initial distribution on states,*
- *E is a set of events,*
- *Ind is a set of functions from S to $\mathbb{R}$ called state indicators (including the constant functions),*
- *$enabled\colon S \to 2^E$ are the enabled events in each state with for all $s \in S$, $enabled(s) \neq \emptyset$.*
- *$delay\colon S \times E \to dist(\mathbb{R}^+)$ is a partial function defined for pairs (s, e) such that $s \in S$ and $e \in enabled(s)$.*

– *choice* : $S \times 2^E \times \mathbb{R}^+ \to dist(E)$ *is a partial function defined for tuples* (s, E', d) *such that* $E' \subseteq enabled(s)$ *and such that the possible outcomes of the corresponding distribution are restricted to* $e \in E'$.
– *target*: $S \times E \times \mathbb{R}^+ \to S$ *is a partial function describing state changes through events defined for tuples* (s, e, d) *such that* $e \in enabled(s)$.

where $dist(A)$ *denotes the set of distributions whose support is* A.

Dynamics of a DESP. A *configuration* of a DESP consists of a triple $(s\ , \tau\ , sched)$ with s being the current state, $\tau \in \mathbb{R}^+$ the current time and $sched : E \to \mathbb{R}^+ \cup \{+\infty\}$ being the function that describes the occurrence time of each scheduled event ($+\infty$ if an event is not yet scheduled). The evolution (i.e simulation) of a DESP $\mathcal{D}$ can be informally summarised in terms of an iterative procedure consisting of the following steps (assuming $(s\ , \tau\ , sched)$ is the current configuration of $\mathcal{D}$): 1) determine the set E_m of events enabled in state s and with minimal delay δ_m; 2) select the next event to occur $e_{next} \in E_m$ by resolving conflicts (if any) between concurrent events through probabilistic choice according to $choice(s, E_m, \tau)$; 3) determine the new configuration of the process resulting from the occurrence of e_{next}, this in turns consists of three sub-steps: 3a) determine the new state resulting from occurrence of e_{next}, i.e. $s' = target(s, e_{next}, \delta_m)$; 3b) update the current time to account for the delay of occurrence of e_{next}, i.e. $\tau = \tau + \delta_m$; 3c) update the schedule of events according to the newly entered state s' (this implies setting the schedule of no longer enabled events to $+\infty$ as well as determining the schedule of newly enabled events by sampling through the corresponding distribution). The above procedure maps directly on GSPN models, in which case the set of states S corresponds to the set of possible markings of a GSPN, the events E correspond to the (timed) transitions of a GSPN, and the remaining elements (i.e. *delay*, *choice* and *target*) are determined by the semantics of GSPN (i.e. the so-called *token game*).

Definition 2. *A* synchronised Linear Hybrid Automaton *is a tuple* $\mathcal{A} = \langle E, L, \Lambda, I, F, X, \mathit{flow}, \to \rangle$ *where:*

– E *is a finite alphabet of events;*
– L *is a finite set of locations;*
– $\Lambda : L \to Prop$ *is a location labelling function;*
– $I \subseteq L$ *is the initial locations;*
– $F \subseteq L$ *is the final locations;*
– $X = (x_1, ...x_n)$ *is a* n*-tuple of data variables;*
– *flow* : $L \mapsto Ind^n$ *associates an* n*-tuple of indicators with each location (projection* flow_i *denotes the flow of change of variable* x_i*).*
– $\to \subseteq L \times \big((\mathsf{Const} \times 2^E) \uplus (\mathsf{lConst} \times \{\sharp\})\big) \times \mathsf{Up} \times L$ *is the set of edges of the LHA*

where $\uplus$ *denotes the disjoint union,* Const *and* lConst *denotes the set of possible* constraints*, respectively* left closed constraints*, associated with* $\mathcal{A}$ *(see description below) and* Up *is the set of possible updates for the variables of* $\mathcal{A}$*. Furthermore* $\mathcal{A}$ *fulfils the following conditions.*

- **c1 (initial determinism):** $\forall l \neq l' \in I$, $\Lambda(l) \wedge \Lambda(l') \Leftrightarrow \texttt{false}$. *This must hold whatever the interpretation of the indicators occurring in* $\Lambda(l)$ *and* $\Lambda(l')$.
- **c2 (determinism on events):** $\forall E_1, E_2 \subseteq E : E_1 \cap E_2 \neq \emptyset$, $\forall l, l', l'' \in L$, *if* $l'' \xrightarrow{\gamma, E_1, U} l$ *and* $l'' \xrightarrow{\gamma', E_2, U'} l'$ *are two distinct transitions, then either* $\Lambda(l) \wedge \Lambda(l') \Leftrightarrow \texttt{false}$ *or* $\gamma \wedge \gamma' \Leftrightarrow \texttt{false}$. *Again this equivalence must hold whatever the interpretation of the indicators occurring in* $\Lambda(l)$, $\Lambda(l')$, γ *and* γ'.
- **c3 (determinism on $\sharp$):** $\forall l, l', l'' \in L$, *if* $l'' \xrightarrow{\gamma, \sharp, U} l$ *and* $l'' \xrightarrow{\gamma', \sharp, U'} l'$ *are two distinct transitions, then either* $\Lambda(l) \wedge \Lambda(l') \Leftrightarrow \texttt{false}$ *or* $\gamma \wedge \gamma' \Leftrightarrow \texttt{false}$.
- **c4 (no $\sharp$-labelled loops):** *For all sequences* $l_0 \xrightarrow{\gamma_0, E_0, U_0} l_1 \xrightarrow{\gamma_1, E_1, U_1} \cdots \xrightarrow{\gamma_{n-1}, E_{n-1}, U_{n-1}} l_n$ *such that* $l_0 = l_n$, *there exists* $i \leq n$ *such that* $E_i \neq \sharp$.

Synchronisation of LHA and DESP. The role of a synchronised LHA $\mathcal{A}$ is to select specific trajectories of a corresponding DESP $\mathcal{D}$ while collecting relevant data (maintained in the LHA variables) along the execution. For the sake of brevity we omit the formal semantics of the product process $\mathcal{D} \times \mathcal{A}$ in this paper, but we provide an intuitive description of it.

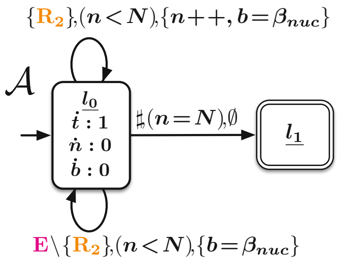
A state of the $\mathcal{D} \times \mathcal{A}$ process is described as a triple (s, l, ν) where s is the current state of the DESP, l the current location of the LHA and $\nu : X \to \mathbb{R}$ the current valuation of the LHA variables. The synchronisation starts from the initial state (s, l, ν), where s is an initial state of the DESP (i.e. $\pi_0(s) > 0$), l is an initial location of the LHA (i.e. $l \in I$) and the LHA variables are all initial set to zero (i.e. $\nu = 0$)[3]. Notice that, by initial determinism, for every $s \in S$ there is at most one $l \in I$ such that s satisfies $\Lambda(l)$. From the initial state the synchronisation process evolves through transitions where each transition corresponds to traversal of either a synchronised or an autonomous edge of the LHA (notice that because of the determinism constraints of the LHA edges at most only one autonomous or synchronised edge can ever be enabled in any location of the LHA. Furthermore if an autonomous and a synchronised edge are concurrently enabled the autonomous transition is taken first). If in the current location of the LHA (i.e. location l of the current state (s, l, ν) of process $\mathcal{D} \times \mathcal{A}$) there exists an enabled autonomous edge $l \xrightarrow{\gamma, \sharp, U} l'$, then that edge will be traversed leading to a new state (s, l', ν') where the DESP state (s) is unchanged whereas the new location l' and the new variables' valuation ν' might differ from l, respectively ν, as a consequence of the edge traversal. On the other hand if an event e (corresponding to transition $s \xrightarrow{e} s'$) triggered by process $\mathcal{D}$ occurs in state (s, l, ν), either an enabled synchronous edge $l \xrightarrow{\gamma, E', U} l'$ (with $e \in E'$) exists leading to new state (s', l', ν') of process $\mathcal{D} \times \mathcal{A}$ (from which the synchronised process will proceed) or the system goes to a dedicated rejecting state $\perp$ and the synchronisation halts (indicating rejection of the trace).

[3] Notice that because of the "initial-nondeterminism" of LHA there can be at most one initial state for the product process.

HASL expressions. The second component of an HASL formula is an expression related to the automaton. Such an expression, denoted Z, is based on moments of a path random variable Y and defined by the grammar (2).

$$\begin{aligned} Z &::= c \mid E[Y] \mid Z+Z \mid Z-Z \mid Z \times Z \mid Z/Z \\ Y &::= c \mid Y+Y \mid Y \times Y \mid Y/Y \mid last(y) \mid min(y) \mid max(y) \mid int(y) \mid avg(y) \\ y &::= c \mid x \mid y+y \mid y \times y \mid y/y \end{aligned} \tag{2}$$

Z represents the expectation of an arithmetic expression based on LHA data variables and which uses path operators such as: $last(y)$ (i.e. the last value of y along a synchronising path), $min(y)$ ($max(y)$) the minimum (maximum), value of y along a synchronising path), $int(y)$ (i.e. the integral over time along a path) and $avg(y)$ (the average value of y along a path). In recent updates the COSMOS model checker [3] has been enriched with operators for assessing the Probability (Cumulative) Distribution Function (PDF/CDF) of the value that an expression Y takes at the end of a synchronising path. This requires specifying a discretised support of Y through the following syntax: $Z = PDF(Y, s, l, h)$ which means that the probability of Y to take value in any sub-interval of fixed width s corresponding to the partition of the considered $[l, h]$ support of Y is going to be evaluated (assuming that $[l, h]$ is discretised in $h - l/s$ sub-intervals).



$\{R_2\},(n<N),\{n{+}{+},b=\beta_{nuc}\}$
$\mathcal{A}$
l_0
$\dot{t}:1$
$\dot{n}:0$
$\dot{b}:0$
$\sharp,(n=N),\emptyset$
l_1
$E\setminus\{R_2\},(n<N),\{b=\beta_{nuc}\}$

Fig. 4. Simple example of LHA that synchronises with the Wnt/β-catenin GSPN model of Figure 1: the automaton selects paths containing N occurrences of reaction R_2

Example Figure 4 shows a simple example of LHA that synchronises with the Wnt/β-catenin GSPN model of Figure 1. It uses three data variables: a clock t (storing the simulation time), a counter n (counting the number of occurrences of reactions R_2) and a variable b which keeps track of the population of β_{nuc}. In the initial location l_0 the clock variable t grows with constant flow $\dot{t}=1$, whereas n and b flows is null. On occurrence of R_2 the top synchronising self-loop edge on l_0 is traversed hence n is incremented whereas on occurrence of any other reaction the bottom self-loop on l_0 is traversed, hence n is not updated. On the hand b is updated with the current value of β_{nuc} on occurrence of any reaction. As soon as N occurrences of R_2 have been observed the autonomous edge $l_0 \to l_1$ is traversed and synchronization halts (reaching of accepting location l_1). Below few examples of complete HASL formulae composed with the LHA of Figure 4.

- $\phi_1 \equiv (\mathcal{A}, E[last(t)])$: representing the average time for observing N occurrences of R_2.
- $\phi_2 \equiv (\mathcal{A}, E[max(b)])$: representing the maximum population reached by β_{nuc} within the first N occurrences of R_2.
- $\phi_3 \equiv (\mathcal{A}, PDF(last(t), 0.1, 0, 10))$: representing the PDF of the delay for observing N occurrences of R_2 (computed over the interval $[0, 10]$ with a discretisation step of 0.1)

4 Model Analysis through HASL Formulae

In order to analyse the dynamics of the Wnt/β-catenin model we define a number of HASL formulae dedicated to capturing specific dynamical aspects of the GSPN model in Figure 1.

4.1 Measuring the Maximal Peaks of β_{nuc} Resulting from an Unsustained Wnt Signal

Both the *Wnt-basic* and *Wnt-inject* models are designed to study the behaviour of the Wnt/β-catenin pathway in presence of an unsustained Wnt signal: i.e. a given amount of initial Wnt signal is present in the system but is steadily being consumed (reaction R_1) without being reintegrated (*Wnt-basic*) or being reintegrated once after a delay d_i (*Wnt-inject*). The effect of a non-reintegrated Wnt signal results is the production of a single peak of β_{nuc} (Figure 2) whereas a single, delayed, reintegration of 1000 Wnt molecules produces a second, shifted peak (Figure 3 left) in the population of β_{nuc}. We introduce some HASL formulae for formally measuring the time location and the amplitude of such β_{nuc} *transient* peaks. Observe that the analysis of a stochastic model through observation of a single simulated trajectory (as proposed in [15]) is in general little informative l and even more so in this case as repeated trajectories of the wntb model exhibit a rather large variance. In the light a more formal approach is vital to obtain a meaningful analysis.

Automaton $\mathcal{A}_{peaks}$. The LHA in Figure 5 is conceived for locating the maximal and minimal peaks along an alternating trace of a given observed species, in this case β_{nuc}. The automaton uses a number of data variables (Table 2) and is dependent on two configuration aspects: the setting of a parameter δ (the chosen noise level, see below) and the partition of the event set $E = E_{+\beta_{nuc}} \cup E_{-\beta_{nuc}} \cup E_{=\beta_{nuc}}$ where $E_{+\beta_{nut}}$, $E_{-\beta_{nuc}}$ and $E_{=\beta_{nut}}$ are the events yielding respectively: an increase of β_{nuc}, a decrease of β_{nuc} and having no effect on β_{nuc} population. Specifically, for model (1) we have: $E_{+\beta_{nuc}} = \{R_{10}\}$, $E_{-\beta_{nuc}} = \{R_{11}\}$ and $E_{=\beta_{nuc}} = \{R_1, R_2, R_3, R_4, R_5, R_6, R_7, R_8, R_9, R_{12}\}$.

$\mathcal{A}_{peaks}$ consists of an initial location **start**, a final location **end**, and 4 intermediate locations (**Min**,**Inc**, **Max** and **Dec**) where the actual analysis of the synchronised trajectory takes place. From **start** (on entering of which the initial amount of β_{nuc} is stored in x) the processing of the simulated trajectory leads either to **Min** or **Max** depending if the an increase (decrease) of β_{nuc} above

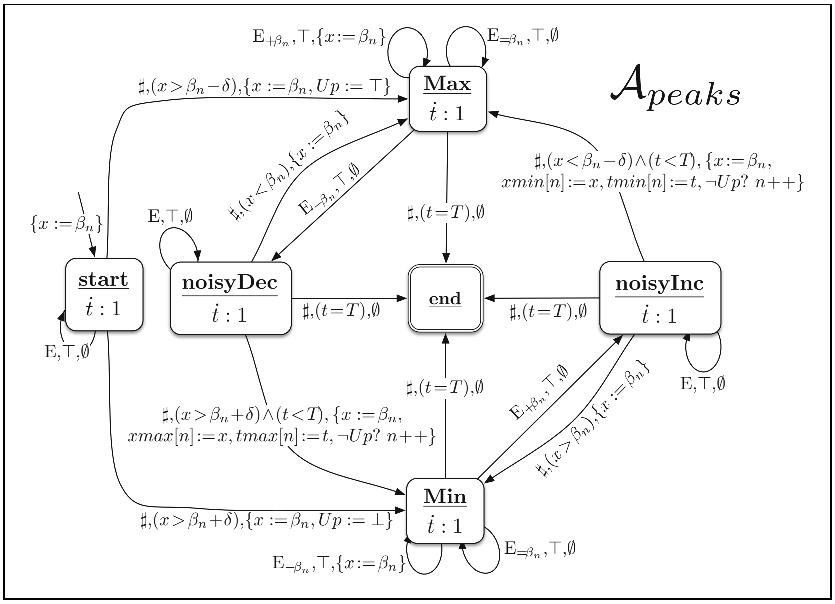

$\mathcal{A}_{peaks}$
Max $\dot{t}:1$
start $\dot{t}:1$
noisyDec $\dot{t}:1$
end
noisyInc $\dot{t}:1$
Min $\dot{t}:1$
$E_{+\beta_n},\top,\{x:=\beta_n\}$
$E_{=\beta_n},\top,\emptyset$
$\sharp,(x>\beta_n-\delta),\{x:=\beta_n, Up:=\top\}$
$\sharp,(x<\beta_n),\{x:=\beta_n\}$
$E_{-\beta_n},\top,\emptyset$
$\sharp,(t=T),\emptyset$
$\sharp,(x<\beta_n-\delta)\wedge(t<T),\{x:=\beta_n, xmin[n]:=x, tmin[n]:=t, \neg Up?\ n{+}{+}\}$
$\{x:=\beta_n\}$
$E,\top,\emptyset$
$\sharp,(x>\beta_n+\delta)\wedge(t<T),\{x:=\beta_n, xmax[n]:=x, tmax[n]:=t, \neg Up?\ n{+}{+}\}$
$E_{+\beta_n},\top,\emptyset$
$\sharp,(x>\beta_n),\{x:=\beta_n\}$
$\sharp,(x>\beta_n+\delta),\{x:=\beta_n, Up:=\bot\}$
$E_{-\beta_n},\top,\{x:=\beta_n\}$
$E_{=\beta_n},\top,\emptyset$

Fig. 5. $\mathcal{A}_{\beta\_peaks}$: an LHA for locating the maximal peaks (up to noise level δ) of β_n

(below) the chosen level of noise δ is observed (i.e. $\mathcal{A}_{peaks}$ copes fine both in the case that the observed species initially increases or decreases).

Once in location **Min** (**Max**) the behaviour of the automaton depends on the type of observed event. If an event $e \in E_{+\beta_{nuc}}$ ($e \in E_{-\beta_{nuc}}$) is observed then location **noisyInc** (**noisyDec**) is entered indicating that β_{nuc} has increased (decreased) although the increase (decrease) has not (yet) exceeded δ (with respect to the most recent detected minimum (maximum) previously stored in x). On the other hand if while in **Min** (**Max**) an event $e \in E_{-A}$ ($e \in E_{+\beta_{nuc}}$) is observed, then this means that the current value of β_{nuc} went below (above) the previously detected minimum (maximum) hence x must be updated with the newly found (potential) minimum (maximum) $x := \beta_{nuc}$. Finally an occurrence of any event $e \in E_{=\beta_{nuc}}$ while in **Min** or **Max** is simply ignored. From **noisyInc** (**noisyDec**) the processing of input trace may lead back to **Min** (**Max**) if β_{nuc} re-decreases (re-increases) below (above) x (hence requiring an update $x := \beta_{nuc}$) or it may lead to **Max** (**Min**)as soon as β_{nuc} has increased (decreased) above (below) the noise level (i.e. $x > \beta_{nuc} - \delta$). The *autonomous* edge **noisyInc**→**Max** (**noisyDec**→**Min**) is traversed as soon as the value stored in x corresponds to an actual minimum (maximum) along the processed β_{nuc} trace (i.e. this is the case when the current value of β_{nuc} gets δ molecules far away from that stored in x). Hence when traversing **noisyInc**→**Max** (**noisyDec**→**Min**) we are sure that x contains a minimum (maximum) thus its value and its occurrence time are stored in the n^{th} element of the array $xmin[n] := x$ ($xmax[n] := x$), respectively $tmin[n] := t$ ($tmax[n] := t$). The processing terminates (entering of **End** from any other location) as soon as the simulation time is $t = T$ at which point all detected maxima and minima are stored in $\mathcal{A}_{peaks}$ data variables.

Table 2. The data variables of automata $\mathcal{A}_{peaks}$ of Figure 5 for locating the peaks of a noisy oscillatory traces

Data variables		
name	domain	description
t	$\mathbb{R}_{\geq 0}$	time elapsed since beginning measure
n	$\mathbb{N}$	counter of detected local maxima/minima
up	bool	boolean flag indicating whether measuring started with detection of a max or min
x	$\mathbb{N}$	(overloaded) variable storing most recent detected maximum/minimum
$xmax(xmin)$	$\mathbb{N}^{\mathbb{N}}$	array of detected maxima (minima)
$tmax(tmin)$	$\mathbb{R}^{\mathbb{N}}$	array of detected occurrence time of maxima (minima)

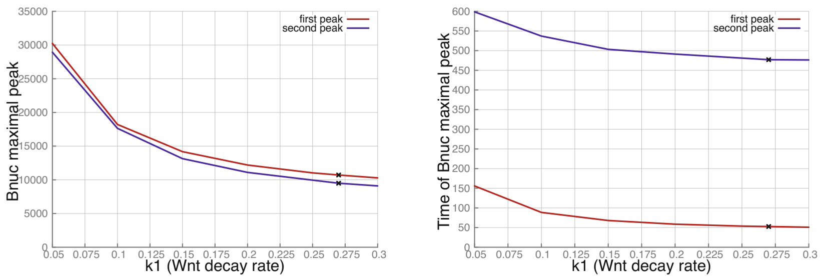


Fig. 6. Average value (left) and occurrence time (right) of the first and second β_{nuc} peak for the *Wnt-inject* model in function of Wnt decay rate (highlighted points correspond to k_1 original value as in Table 1 set B)

HASL formulae for measuring the effects of unsustained Wnt signal. Based on automaton $\mathcal{A}_{peaks}$ we define the following HASL formulae:

- $\phi_{xmax} \equiv (\mathcal{A}_{peaks}, E[last(xmax[1])])$: the average value of the first β_{nuc} maximal peak.
- $\phi_{tmax} \equiv (\mathcal{A}_{peaks}, E[last(tmax[1])])$: the average value of the occurrence time of first β_{nuc} maximal peak.
- $\phi_{PDFmax} \equiv (\mathcal{A}_{peaks}, PDF(last(tmax[1]), 1, 30, 80))$: the PDF of the occurrence time of first β_{nuc} maximal peak (computed over the interval $[30, 80]$ with a discretisation step 1)

Measuring the incidence of Wnt decay rate on β_{nuc} peaks. The decay speed of the Wnt signal affects the dynamics of β_{nuc} (the temporal location and height of β_{nuc} peaks). We performed a number of experiments aimed at addressing this aspect, specifically we assessed ϕ_{xmax} and ϕ_{tmax} against different instances of the Wnt/β-catenin model with delayed Wnt re-injection (i.e. the *Wnt-inject* model) where each instance corresponds to a different value of k_1 (the decay rate Wnt). Figure 6 displays the results concerning the evaluation of the first and second

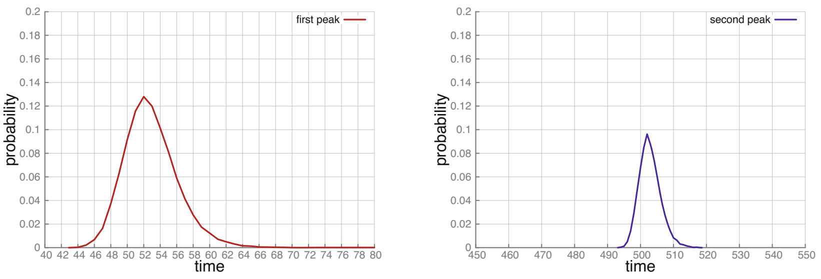


Fig. 7. The PDF of the time of occurrence of the first β_{nuc} peak (left) and second peak (right) for the *Wnt-inject* model (with re-injection of 1000 Wnt molecules at $t = 450$ minutes)

peak of β_{nuc}. They indicate that both the average height (left) and the average occurrence time (right) of the first and second peaks of β_{nuc} decrease as the Wnt decay rate k_1 increases[4]. Figure 6 left also shows that the second peak of β_{nuc} (induced by Wnt re-injection) has, on average, a smaller amplitude than the first one and with a roughly constant difference of about 10% less between the two except for a $k_1 = 0.1$ for which the first and second peak's amplitude differs of about 5%.

Measuring the PDF of occurrence time of β_{nuc} *peaks.* Figure 7 displays the PDF of the first (left) and second (right) peak of β_{nuc} obtained by evaluation of ϕ_{PDFmax} against the *Wnt-inject* model (parameter set B). Both PDF curves exhibit a slight long-tail character with the majority of points being to the right of the maximum likely occurrence time.

5 Conclusion

We have presented a formal study of a stochastic model of the Wnt/β-catenin pathway, a biological mechanism with a relevant role in controlling the life-cycle of neuronal embryonal cells. This model has been previously considered [15] however it was analysed only *informally*, i.e. through observation of simulated trajectories. By means of a powerful formalism (i.e. HASL model checking) we formally characterised and accurately assessed a number of relevant aspects of the Wnt/β-catenin dynamics. In particular in this work we have focused on studying the effects induces on nuclear β-catenin (a basic element of the Wnt/β-catenin pathway) by the presence of a degrading (possibly reintegrated) Wnt signal. That included measuring of the average value and the PDF of the occurrence time and the amplitude of the β_{nuc} peaks resulting from a transitory Wnt signal. We plan to evolve this preliminary study in several directions, including the formal analysis of the effects

[4] Results computed with confidence level 99% and interval-width of 1% of the estimated measure.

induced by a sustained Wnt signal, as well as the analysis of the dynamics of β_{nuc} over a population of asynchronously evolving cells.

References

1. Ajmone Marsan, M., Balbo, G., Conte, G., Donatelli, S., Franceschinis, G.: Modelling with Generalized Stochastic Petri Nets. John Wiley & Sons (1995)
2. Baier, C., Haverkort, B., Hermanns, H., Katoen, J.-P.: Model-checking algorithms for CTMCs. IEEE Trans. on Software Eng. 29(6) (2003)
3. Ballarini, P., Djafri, H., Duflot, M., Haddad, S., Pekergin, N.: COSMOS: a statistical model checker for the hybrid automata stochastic logic. In: Proceedings of the 8th International Conference on Quantitative Evaluation of Systems (QEST 2011), pp. 143–144. IEEE Computer Society Press (September 2011)
4. Ballarini, P., Djafri, H., Duflot, M., Haddad, S., Pekergin, N.: HASL: an expressive language for statistical verification of stochastic models. In: Proc. Valuetools (2011)
5. Ballarini, P., Mäkelä, J., Ribeiro, A.S.: Expressive statistical model checking of genetic networks with delayed stochastic dynamics. In: Gilbert, D., Heiner, M. (eds.) CMSB 2012. LNCS, vol. 7605, pp. 29–48. Springer, Heidelberg (2012)
6. Boyer, B., Corre, K., Legay, A., Sedwards, S.: PLASMA-lab: A flexible, distributable statistical model checking library. In: Joshi, K., Siegle, M., Stoelinga, M., D'Argenio, P.R. (eds.) QEST 2013. LNCS, vol. 8054, pp. 160–164. Springer, Heidelberg (2013)
7. Chen, T., Han, T., Katoen, J.-P., Mereacre, A.: Quantitative model checking of CTMC against timed automata specifications. In: Proc. LICS 2009 (2009)
8. David, A., Larsen, K.G., Legay, A., Mikučionis, M., Poulsen, D.B., Sedwards, S.: Runtime verification of biological systems. In: Margaria, T., Steffen, B. (eds.) ISoLA 2012, Part I. LNCS, vol. 7609, pp. 388–404. Springer, Heidelberg (2012)
9. Donatelli, S., Haddad, S., Sproston, J.: Model checking timed and stochastic properties with CSL^{TA}. IEEE Trans. on Software Eng. 35 (2009)
10. Heath, J., Kwiatkowska, M., Norman, G., Parker, D., Tymchyshyn, O.: Probabilistic model checking of complex biological pathways. Theoretical Computer Science 319(3), 239–257 (2008)
11. Kitano, H.: Foundations of Systems Biology. MIT Press (2002)
12. Kwiatkowska, M., Norman, G., Parker, D.: Stochastic model checking. In: Bernardo, M., Hillston, J. (eds.) SFM 2007. LNCS, vol. 4486, pp. 220–270. Springer, Heidelberg (2007)
13. Legay, A., Delahaye, B.: Statistical model checking: An overview. CoRR, abs/1005.1327 (2010)
14. Mazemondet, O., Hubner, R., Frahm, J., Koczan, D., Bader, B.M., Weiss, D.G., Uhrmacher, A.M., Frech, M.J., Rolfs, A., Luo, J.: Quantitative and kinetic profile of wnt/β-catenin signaling components during human neural progenitor cell differentiation. Cell. Mol. Biol. Lett. (2011)
15. Mazemondet, O., John, M., Leye, S., Rolfs, A., Uhrmacher, A.M.: Elucidating the sources of β-catenin dynamics in human neural progenitor cells. PLOS-One 7(8), 1–12 (2012)
16. Sen, K., Viswanathan, M., Agha, G.: VESTA: A statistical model-checker and analyzer for probabilistic systems. In: Proc. QEST 2005 (2005)
17. Younes, H.L.S.: Ymer: A statistical model checker. In: Etessami, K., Rajamani, S.K. (eds.) CAV 2005. LNCS, vol. 3576, pp. 429–433. Springer, Heidelberg (2005)

Battery-Aware Scheduling of Mixed Criticality Systems[*]

Erik Ramsgaard Wognsen, René Rydhof Hansen, and Kim Guldstrand Larsen

Department of Computer Science
Aalborg University, Denmark
{erw,rrh,kgl}@cs.aau.dk

Abstract. Wireless systems such as satellites and sensor networks are often battery-powered. To operate optimally they must therefore take the performance properties of real batteries into account. Additionally, these systems, and therefore their batteries, are often exposed to loads with uncertain timings. Mixed criticality and soft real-time systems may accept deadline violations and therefore enable trade-offs and evaluation of performance by criteria such as the number of tasks that can be completed with a given battery. We model a task set in combination with the kinetic battery model as a stochastic hybrid system and study its performance under battery-aware scheduling strategies. We believe that this evaluation does not scale with current verification techniques for stochastic hybrid systems. Instead statistical model checking provides a viable alternative with statistical guarantees. Based on our model we also calculate an upper bound on the attainable number of task instances from a battery, and we provide a battery-aware scheduler that wastes no energy on instances that are not guaranteed to make their deadlines.

1 Introduction

By their very nature, embedded systems often have to operate independently, powered only by a battery that may or may not be reliably recharged by an external power source. As an example of this consider a satellite with solar panels used to recharge the on-board battery. Since the task load for such a satellite may vary depending on the current position and activity of the satellite, it can be difficult to precisely predict energy consumption and, consequently, whether the mission can be accomplished within the current energy budget. In this paper we model task loads in combination with the *kinetic battery model*. This enables formal (statistical) modeling and verification of energy consumption and availability, as well as battery state, over extended periods of time for non-trivial task sets.

For our task model, we want to take into account the timing characteristics, in terms of arrival periods and patterns, as well as the costs (execution times) of

[*] This work is partially supported by SENSATION (Self Energy-Supporting Autonomous Computation) under the EU's Seventh Framework Programme.

the tasks. For such modeling, *timed automata* [5] have been proposed and successfully used as a modeling formalism. Furthermore, various extensions of timed automata have been proposed, e.g., for schedulability analysis [23]. Similarly, another extension of timed automata, namely *(linearly) priced timed automata* [7], has been demonstrated to be a suitable formalism for modeling batteries [19]. In particular, this formalism allows for both optimal reachability and optimal infinite runs to be computed in PSPACE [9,10]. However, both the timed automata model of tasks and the priced timed automata of batteries will be only approximate and leave out important details. More exact battery models, such as the kinetic battery model [21], will require the use of hybrid automata as modeling formalism. Also, in modeling of tasks the arrival pattern and execution times may be refined into stochastic information about arrivals and distributions of the execution times. In both cases, we will use the notion of stochastic hybrid systems [11] to allow for more precise models. This enables us to use statistical model checking [13] to study system performance under various soft real-time scheduling principles.

2 Related Work

The kinetic battery model, KiBaM, may be directly represented as a hybrid automaton [17], where the evolution of the levels of "available charge" and "bound charge" are described by a system of ordinary differential equations depending on the current load on the battery.

In case the dynamic load of the battery may itself be described as a hybrid automaton, properties of the overall system (e.g. estimation of battery life-time) should in principle be amenable to verification using the various abstractions techniques – e.g. grid partitioning – devised for hybrid systems [3] and with the assistance of popular modeling and verification systems such as model checking CHARON, PHAVer, HSolver, d/dt, and CheckMate.

Aiming at more abstract battery models in terms of linear hybrid automata, timed automata or priced timed automata may allow formal verification (or even optimization as in [19]) to be performed using popular model checkers such as HyTech, and UPPAAL or UPPAAL Cora. For a more in-depth account of the approaches to formal verification of hybrid systems we refer to [4].

Here we want to consider *stochastic* models of the load of the battery giving rise to an overall stochastic hybrid automata model [8]. A number of such formalisms has been proposed varying with respect to the actual placement of stochastic aspects (at discrete jumps, in the determination of the time of discrete jumps or during the continuous evolution of continuous variables). In terms of (approximate) verification of stochastic hybrid systems, we mention the work by [25,1,2], and the recently released hybrid version of the stochastic modeling and analysis tool MODEST, HMODEST [16].

In this paper, we will be using the recently developed extension of the modeling formalism of UPPAAL supporting (networks of) stochastic hybrid systems [11]. Rather than supporting formal verification, the new UPPAAL SMC engine performs highly scalable, statistical model checking, which allows the probabilities and expected values of a large set of path-based properties and random variables to be determined not exactly but with a given desired level of confidence. We believe that for evaluating the performance of various (battery-) scheduling strategies, current verification techniques for stochastic hybrid systems will not scale. Related to our approach is the statistical model checking of stochastic hybrid systems presented in terms of Stateflow/Simulink models in [22,26].

3 Battery Modeling

Batteries consist of electrochemical cells in which electrochemical reactions transform chemical energy to electrical energy. Battery modeling can be used to predict, among other things, the battery voltage and state of charge during the run of a battery-powered system. In this paper we restrict our attention to the state of charge. This is reasonable since the voltage varies considerably less than the state of charge, and many systems will already employ voltage regulation to ensure stable a voltage.

Jongerden and Haverkort survey various battery models in [18]. Electrochemical models are the typical reference models which most closely model the electrochemical reactions at the cost of a high complexity and dependence on numerous physical and chemical attributes of the electrochemical cells. Another approach is found in electrical circuit models which abstract away the chemical reactions and model batteries as combinations of electrical components such as capacitors and resistors. These models are computationally cheaper than electrochemical models but they are less accurate and depend on a large amount of experimental data to configure. Analytical models further abstract the details of the reactions. The kinetic battery model by Manwell and McGowan [21] considers only the rates of the chemical reactions and partitions the charge in the battery into available and (temporarily) bound charge. A more advanced analytical model, the diffusion model by Rakhmatov and Vrudhula, considers the concentration of reactants as a continuous gradient between the two electrodes in the battery. The authors of [18] find analytic models to be the most well suited for performance modeling and combination with workload models. Specifically the kinetic battery model has the best accuracy for practical purposes compared to its complexity.

3.1 The Kinetic Battery Model

The kinetic battery model [21] partitions the charge into two "wells" as illustrated in Fig. 1. The outer well, termed the *available charge*, powers the load while the inner well, the *bound charge*, replenishes the outer well. While the water in the wells represents electric charge it should not be taken as a liquid electrolyte, and the wells do not represent the physical layout of an electrochemical cell. Charge leaves the battery at the time-dependent rate i. The widths of

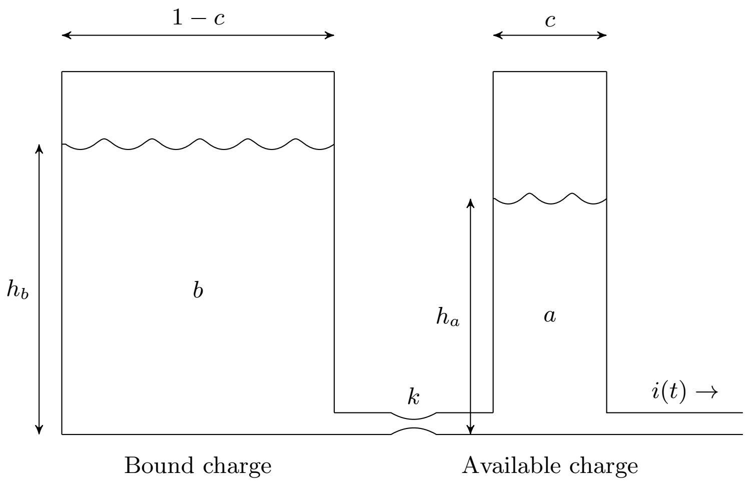


Fig. 1. Mental model of the kinetic battery model

the available and bound charge wells are c and $1-c$, respectively. With the wells filled to heights h_a and h_b the charges in the wells are $a = ch_a$ and $b = (1-c)h_b$. The charge flows between the wells at a rate proportional to the height difference with the proportionality factor being the rate constant k. The charges are thus described by this system of differential equations:

$$\begin{cases} \frac{\mathrm{d}a}{\mathrm{d}t} = -i + k(h_b - h_a) = -i + k(\frac{b}{1-c} - \frac{a}{c}) \\ \frac{\mathrm{d}b}{\mathrm{d}t} = -k(h_b - h_a) = -k(\frac{b}{1-c} - \frac{a}{c}) \end{cases} \tag{1}$$

All variables except c and k may change over time. The system starts in equilibrium, $h_a = h_b$, so with an initial capacity of C we have $a(0) = cC$ and $b(0) = (1-c)C$. The battery is considered empty when $a = h_a = 0$ since it cannot supply any more charge at the given moment even though it may still contain bound charge. In fact, due to the dynamics of the system, the bound charge cannot reach 0 in finite time. The parameters c and k depend on the battery technology and can be fitted experimentally but in general the model applies to all chemical batteries. In our experiments we use $c = \frac{1}{6}$ and $k = 2.324 \times 10^{-4}\ \mathrm{s}^{-1}$, similar to the ones used in [19].

The equation system can be solved analytically for a known (piecewise) constant load I. In particular, the available charge a after the current I has been drawn for time t is calculated as

$$a(t, I, a_0, b_0) = a_0 e^{-k't} + \frac{(q_0 k' c - I)(1 - e^{-k't}) - Ic(k't - 1 + e^{-k't})}{k'} \tag{2}$$

where a_0 and b_0 make up the battery state at relative time $t = 0$, $q_0 = a_0 + b_0$, and $k' = \frac{k}{c(1-c)}$. We will exploit this later in the development of a battery-aware scheduling principle.

3.2 Comparison with an Ideal Energy Source

Much work concerning costs, resources, and energy use an ideal energy source which is simply a scalar variable that can be changed at will. As long as its value is positive the system is considered to be healthy, energy-wise. With an explicit representation of time we describe the energy e as

$$\frac{\mathrm{d}e}{\mathrm{d}t} = -i$$

To illustrate the (large) difference between ideal energy sources and real batteries, we compare the two in Figs. 2 and 3. The former shows a UPPAAL simulation using an ideal energy source (see Section 4.1) powering a periodic load with a duty cycle of 50%. The latter shows the same load powered by a (KiBaM) battery of the same total capacity. We see that conclusions on system performance may be catastrophically wrong if a battery is modeled as an ideal energy source.

4 System Modeling

UPPAAL SMC supports the analysis of stochastic hybrid automata (SHA) that are timed automata whose clock rates can be changed to be constants or expressions depending on other clocks, effectively defining ODEs [11,12,20]. This generalizes the formalism used in previous work [15,14] where only linear priced automata were handled. The release UPPAAL 4.1.18[1] supports fully hybrid automata with ODEs and a few built-in complex functions (such as $sin, cos, log, exp, sqrt$).

A UPPAAL model consist of a network of timed automata. Each component automaton consists of locations and edges and the components synchronize over communication channels. We first present the UPPAAL model underlying the ideal energy source example in Fig. 2.

4.1 Ideal Energy Source

The model is comprised of the three components shown in Fig. 4 as determined by the system declaration "**`system`** `Task, EnergySource, Scheduler;`".

The starting location of each component is marked with a concentric circle, and a rounded 'U' marks an urgent location, i.e., one wherein time cannot pass. Locations can be annotated with names and invariants. Names (e.g., Ready) have no effect on system behavior. Invariants (e.g., `t <= p`) are conditions that

[1] `www.uppaal.org`

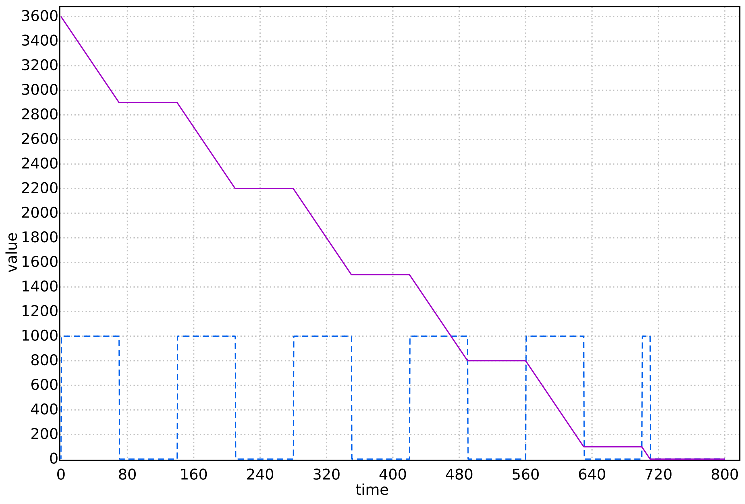


Fig. 2. Simulation of an ideal energy source (solid line, unit Coulombs) during a periodic load (dashed line, unit milliamperes). The energy runs out just before time 720.

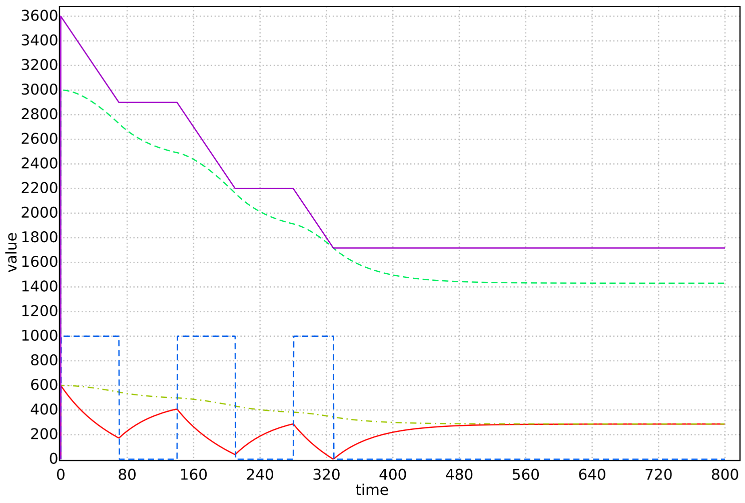
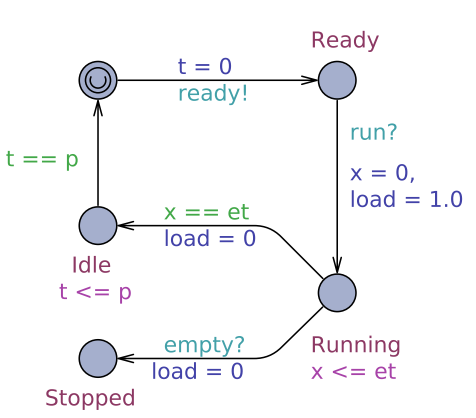
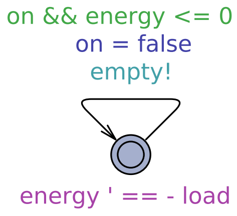


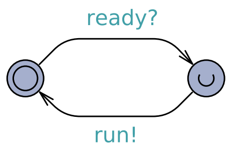
Fig. 3. Simulation of the same periodic load as in Fig. 2 powered by a battery with the same capacity. The upper solid line represents the total charge in the battery, i.e., the sum of the bound charge (b, the upper dashed line) and the available charge (a, the lower solid line). The alternating dotted/dashed line represents the height h_b scaled so it can be compared to the available charge as if it were h_a (in reality, $h_a = \frac{a}{c} = 6a$ and $h_b = \frac{b}{1-c} = \frac{6b}{5}$, so we scale both by $\frac{1}{6}$ so a and h_a coincide). Just after time 320 the available charge runs out and the system fails even though the battery has expended only a good of half its charge. A system with more lenient requirements would be able to wait for more charge to become available and thus exploit more of the total charge.



(a) Basic task model



(b) Ideal energy source



(c) Scheduler

Fig. 4. UPPAAL models for the ideal energy source example

must hold for the component to be in that location. These conditions range over clocks and variables which are explained below. Invariants can also specify rates of growth for clocks (with a prime, e.g., `energy' == -load`). In this way, clocks can be exploited as continuous variables, with differential equations in invariants describing their dynamics. Clocks with unspecified rates have rate one.

Edges can be annotated with guards, synchronization (explained below), and updates. Guards (e.g., `x == et`) are conditions that must hold to take the transition represented by the edge. Updates (e.g. `t = 0`) change discrete variables and clocks. The data declarations for the current example are:

```
clock energy = 3600.0;  bool on = true;  // in coulombs, status
const int et = 700, p = 1400;  // task execution time and period
const double load_on = 1.0;  // active load in amperes
clock t, x;  // time since release, accumulated execution time
double load;  // instantaneous load in amperes
broadcast chan ready, run;  urgent broadcast chan empty;
```

UPPAAL SMC uses broadcast synchronization between components which means that sending (e.g., `ready!`) never blocks the sending component. Any matching receiver (`ready?`) that is in a position to follow the synchronization does so, and all involved components take their transitions simultaneously.

The simulation in Fig. 2 of this system is made with the query

```
simulate 1 [<=8000] {load*1000, energy}
```

In the sequel we consider other versions of the presented system components, starting with the energy source in the next section, the task model in Section 5, and the scheduler in Section 7.

4.2 UPPAAL KiBaM

The chemical reaction in a battery is a continuous process of convergence towards equilibrium described by (1). To model this as a stochastic hybrid automaton, we need only a single location, with (1) given as its invariant. The model is shown in Fig. 5 and based on the following declarations, here with some sample values of C, c, and k:

```
const double C = 3600.0, c = 1.0/6, k = 2.324e-4, k2 = k/c/(1-c);
clock a = c*C, b = (1-c)*C;  bool on = true;
```

As in Fig. 4b we use a self-loop is to signal to the load model when the battery is empty and to “disconnect” the battery.

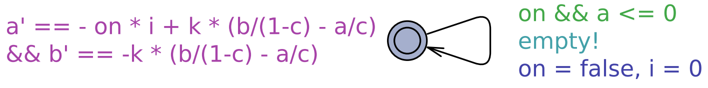

a' == - on * i + k * (b/(1-c) - a/c)
&& b' == -k * (b/(1-c) - a/c)
on && a <= 0
empty!
on = false, i = 0

Fig. 5. UPPAAL kinetic battery model

While this model cannot be used in non-hybrid model checking, it is a much simpler and more obviously correct model than the explicitly discretized version of KiBaM described by timed automata in [19]. That version of KiBaM consists of two automata with 10 locations and 16 transition in total.

5 Mixed Criticality Systems

Battery-aware scheduling will contribute the most to systems with varying energy consumption since it allows the battery to recover capacity between high loads. Consider for example an energy-constrained satellite with an on-board computer, a radio, and a payload. The on-board computer(s) run relatively frequent tasks while the radio communicates in energy-intensive bursts when the satellite passes close to ground stations. The payload could be either a spread out load as the computer or a more “clumped” load as the radio. The on-board computer manages the battery and receives commands from the ground station and must always run. With the satellite being semi-autonomous it may however continue operating while energy conditions are such that the radio or payload (temporarily) cannot be used.

We consider mixed criticality systems, like the one described, where the computer will run hard real-time control tasks while the radio and payload will be considered firm real-time tasks in the sense that a task instance that does not complete within its deadline contributes no value to the operation of the system. It is thus allowed to skip task instances (but not “too many”). Taking this perspective allows trade-offs between missing some deadlines and other aspects of the system performance, notably the energy. On the other hand, the simplest way to save energy is to skip each and every task instance so there must be an additional quality-of-service requirement.

5.1 Task Model

The speed of the diffusion of reactants in an electrochemical cell gives rise to a low-pass filtering of the changes in load. Since the speed of the diffusion is relatively slow there will be very little to gain from battery-aware scheduling at the scale of milliseconds. From the perspective of the battery we consider the frequent computer-based tasks to make up a baseline load of the system called the idle load. Tasks within this group are not explicitly scheduled since they should never be skipped and their timings are typically on a scale that is too small to be distinguished from a constant load by the battery. In other words, they are assumed to be scheduled by a traditional (battery-unaware) scheduler on a platform with adequate resources.

Thus we consider scheduling of "high-level" tasks with running times in the scale of minutes on top of a constant idle load. Such high-level tasks may for example represent an ongoing process such as radio transmission of data or attitude control, i.e., the spatial orientation of a satellite which may involve periodical sensing and actuation over a span of minutes while rotational oscillation is dampened.

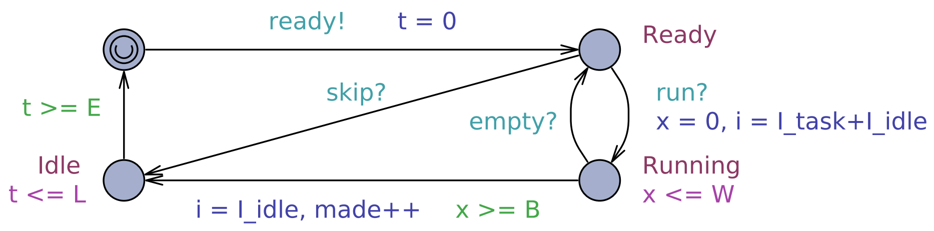


Fig. 6. Firm stochastic task model

Fig. 6 shows the task model with its `skip` transition and counting of the deadlines `made`. The model is parameterised by the best-case execution time B, worst-case execution time W, deadline D, and earliest and latest release times E and L (where $B \leq W \leq D \leq E \leq L$), as well as the loads I_{task} and I_{idle}. We are going to consider schedulers that will either skip each instance or start it in time to finish before its deadline, battery allowing. For traditional schedulability analysis, the model could be extended with deadline violations leading to an error location but for now this will complicate the presentation unnecessarily.

The workload is stochastic when $B < W$ or $E < L$. In UPPAAL SMC uniform probability distributions are applied for bounded delays such as these (unbounded delays use an exponential distribution). In a location where a stochastic choice must be made, the component chooses a delay independently of the rest of the system.

Any empirically determined or otherwise desired distribution can be approximated by a phase-type distribution [24] which can be incorporated into the model.

6 Bound on Performance

We consider battery-aware scheduling where the challenge in energy constrained systems lies: In not having enough energy. While satellites typically recharge their batteries via solar panels, they will need to survive their time on the dark side of the planet, or handle situations where solar panels fail. And even for systems with conservative power budgets this may help system designers be less pessimistic and utilize a larger portion of their chosen battery or alternatively replace the battery with a smaller, cheaper, or lighter model.

The typical battery situation that is desirable to put off for as long as possible is running out of energy. But other thresholds can be considered too. For example, a system may enter a low-power mode at a 20% state of charge but it is desirable to operate in the normal mode for as long as possible. While any such threshold can be considered we assume running out of energy as the demonstration case.

To get an idea for the improvement a battery-aware scheduler could contribute we consider how many task instances a given battery can deliver in the best case. If we optimistically view a battery of capacity C as an ideal energy source, we can in the best case finish

$$UB_{ideal} = \left\lfloor \frac{C}{IB} \right\rfloor \tag{3}$$

whole task instances that each draw the constant current I for at least the time B. Here I is the total load on the battery, i.e., $I = I_{task} + I_{idle}$. If $I_{idle} > 0$, UB_{ideal} is not necessarily tight since there is a load on the battery between task instances. With the shortest task period (E) known, a tight upper bound under the ideal energy model could be determined. But even this would not (necessarily) be a tight upper bound under KiBaM, so we will look to that for a more realistic bound.

As the bound charge in the battery falls during usage the largest possible value of the quantity $h_b - h_a$ falls, and with it, the rate of flow from the bound to the available charge. For a task instance to complete, the necessary charge must be or become available in the duration of that same instance. At the end of the battery life this replenishment can happen too slowly to allow enough charge to be released while the instance is executing.

Using (2) we can calculate the most depleted battery state that can finish exactly one more task instance in the best case before running dry. Working backwards from that state we can determine how many instances could be completed before reaching it. We want to obtain the a_0 and b_0 that satisfy

$$a(B, I, a_0, b_0) = 0 \tag{4}$$

In the best case, the battery would be in equilibrium before running the last instance. Using $h_a = h_b \Leftrightarrow a/c = b/(1-c) \Leftrightarrow b = ((1-c)a)/c$ we get

$$q_0 = a_0 + b_0 = a_0 + \frac{(1-c)a_0}{c} = \frac{a_0}{c} \tag{5}$$

Substituting (2) into (4) and using (5) we get the total charge in the battery

$$q_0 = \frac{I}{k'c}(1 + (k'B - 1)c + (c - 1)e^{-k'B}) \tag{6}$$

Under appropriate scheduling, i.e. enough time for replenishment, the first $C - q_0$ of charge can power as many (say, m) instances as an ideal energy source, cf. (3). The remainder q_0 can power exactly one more instance. However, since the equilibrium can only be reached in the limit, we require the execution of the m instances to leave behind strictly greater than q_0 total charge such that it is possible in finite time to release enough available charge to complete the last instance. To account for this, instead of rounding down (as in (3)) and adding one, we round up:

$$UB_{KiBaM} = \left\lceil \frac{C - q_0}{IB} \right\rceil \tag{7}$$

This bound is closer than UB_{ideal}. But as before, if $I_{idle} > 0$, it is not guaranteed that this number of instances can be reached. Another system requirement is likely to interfere as well: Quality-of-service, i.e., a specification on the least number of instances that must meet their deadline. Taking these facts into the calculation of a tight bound becomes more involved. Instead we turn to statistical model checking to evaluate performance of actual scheduling principles.

7 Evaluation of Scheduling Principles

As a basis for evaluating energy-aware schedulers we consider the simple, battery-unaware immediate scheduler, shown in Fig. 4c, which starts task instances as soon as they are released and never skips them. We will compare this with the battery-aware "firm scheduler".

The charge that leaves the battery over time fully determines the state of the battery[2]. Since this charge is electrically observable, we assume that a scheduler can know the internal state of the battery, i.e., the values of the bound and available charge.

7.1 Firm Scheduler

When a task instance is released, the firm scheduler calculates the worst battery state that the execution of the task could result in, i.e., the task's load drawn from the battery for the worst-case execution time according to (2). If this state has any available charge left, the instance is started immediately. If the calculated state has a non-positive available charge, the scheduler awaits the battery state improving (bound charge becoming available) such that the available charge would remain positive after executing the task. If the battery state does not

[2] Assuming KiBaM adequately models the battery. In practice the age and temperature of the battery as well as measurement accuracy may affect the value of this technique.

improve enough while there would still be enough time for the instance to finish before its deadline, the instance is skipped, and no energy is wasted trying to make it to the deadline. Fig. 7 shows the scheduler. Compared with the example in Section 4.1 `run` is now an urgent channel. The `firm_thresh` value, set to 0.1 in our experiments, allows the scheduler to err on the side of caution in the face of rounding errors in the hybrid system discretization.

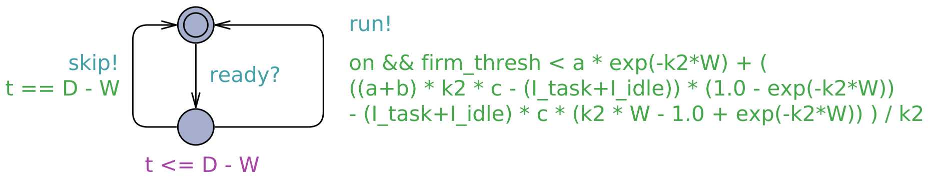

skip!
t == D - W
ready?
run!
on && firm_thresh < a * exp(-k2*W) + (
((a+b) * k2 * c - (I_task+I_idle)) * (1.0 - exp(-k2*W))
- (I_task+I_idle) * c * (k2 * W - 1.0 + exp(-k2*W))) / k2
t <= D - W

Fig. 7. The firm scheduler. The guard on the right hand side uses (2).

7.2 Comparison

We evaluate the schedulers on a system with the same battery as in Section 4.2, and a task set with $B = 100, W = 200, D = 300, E = 300, L = 400, I_{task} = 0.5, I_{idle} = 0.02$.

As mentioned in Section 5.1, these values give rise to stochastic behavior. The fundamental principle in UPPAAL SMC is to generate runs and evaluate some expression on the states along the obtained run. Runs are always *bounded*, either by time, by a number of steps, or more generally by cost (when using a clock explicitly). The engine supports probability evaluation, hypothesis testing, probability comparison, and value estimation, as well as the simulations previously seen.

Putting a copy of the battery and task model in parallel with each of the two schedulers, we can compare them. First we use value estimation and record the maximal number of task instances completed for the two schedulers over a specified number of runs, here 1000:

```
E[<=28800; 1000] (max: Task(0).made)
E[<=28800; 1000] (max: Task(1).made)
```

`Task(0)` is controlled by the immediate scheduler and `Task(1)` by the firm scheduler. The time bound is set such that even with the task taking its best-case execution time each time, the battery will be used up. While time is abstract in UPPAAL, our model is based on real-world values such that one time unit corresponds to one second. Fig. 8 shows the observed probabilities of completing the shown numbers of instances under the two schedulers.

The number 1000 was chosen arbitrarily. To get results with statistical guarantees we can for example ask the following query: What is the probability that the immediate scheduler will finish at least 36 instances within eight hours?

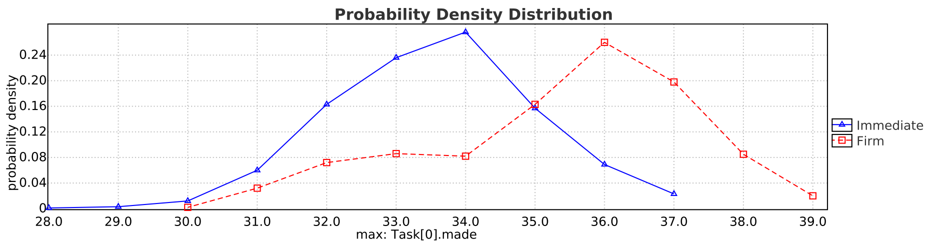


Fig. 8. Comparison of the schedulers using value estimation

```
Pr[<=28800] (<> Task(0).made >= 36)
```

UPPAAL answers that the probability is in the interval $[0.0504475, 0.10035]$ with confidence level 0.95. The result was determined using 455 runs. The probability uncertainty (here, $\varepsilon = 0.025$) and significance level (here, $\alpha = 0.05$) are configurable. We can also ask the tool to perform probability comparison using sequential testing:

```
Pr[<=28800] (<> Task(1).made>=36) >= Pr[<=28800] (<> Task(0).made>=36)
```

With this method, the two probabilities are compared without estimating them directly and it is thus more efficient. For this query 148 runs were enough to determine that with confidence level 0.95, the ratio of the two probabilities is greater than 1.1 (with the extra 0.1 being a safety margin). In other words, the firm scheduler is better, as expected from Fig. 8.

7.3 Variation on the Firm Scheduler

It may turn out that the firm scheduler as shown is too pessimistic since it only starts a task instance that is guaranteed to be able to finish. Perhaps by using a value smaller than W the overall performance may be improved. We introduce the parameter p and instead of $t = W$ in (2) we use $t = B + p(W - B)$.

Using probability comparison we get that with $p = 0.7$, it is less likely to complete 36 instances than with $p = 1$ (with confidence level 0.95, the ratio is less than or equal to 0.9 after 374 runs). On the other hand, with $p = 0.7$ it is *more* likely to finish 40 instances (with confidence level 0.95, the ratio is greater than or equal to 1.1 after 1136 runs) showing that the choice of the best strategy is not completely straightforward.

8 Conclusion

In this paper we have shown the applicability of statistical model checking to battery-powered systems. The kinetic battery model has been modeled as a hybrid system in the UPPAAL formalism in a straightforward way, which has the advantage that it is easy to see its correctness. We combine the battery with a high-level mixed criticality real-time task model. Based on the models, we determine a bound on the performance of a battery-aware scheduler measured as the number of (non-critical) deadlines that are made. We present a scheduling principle that takes the inner workings of the kinetic battery model into account by wasting no energy on task instances that are not guaranteed to make their deadlines. A variation on this relaxes the guarantee while making it more likely to complete an especially high number of task instances at the cost of a lower probability of achieving a moderate number of instances.

Future work includes considering the energy harvesting and the recharging of batteries. Here the challenge shifts from making a single battery charge last for as long as possible to prolonging the total lifetime of the rechargeable battery with respect to reduced capacity. The number and depth of discharge/recharge cycles as well as the environment temperature affects the total lifetime. Other challenges include dynamic mission re-planning after changes in the power budget, e.g., as a consequence of a solar cell damage. Another avenue involves using SMC to approximate optimal scheduling strategies as a scalable alternative to exact optimization.

References

1. Abate, A., Katoen, J.P., Lygeros, J., Prandini, M.: Approximate model checking of stochastic hybrid systems. European Journal of Control 16, 624–641 (2010), `http://control.ee.ethz.ch/index.cgi?page=publications;action=details;id=3711`
2. Abate, A., Prandini, M., Lygeros, J., Sastry, S.: Probabilistic reachability and safety for controlled discrete time stochastic hybrid systems. Automatica 44(11), 2724–2734 (2008), `http://dx.doi.org/10.1016/j.automatica.2008.03.027`
3. Alur, R., Courcoubetis, C., Halbwachs, N., Henzinger, T.A., Ho, P.H., Nicollin, X., Olivero, A., Sifakis, J., Yovine, S.: The algorithmic analysis of hybrid systems. Theor. Comput. Sci. 138(1), 3–34 (1995)
4. Alur, R.: Formal verification of hybrid systems. In: Proceedings of the Ninth ACM International Conference on Embedded Software, EMSOFT 2011, pp. 273–278. ACM, New York (2011), `http://doi.acm.org/10.1145/2038642.2038685`
5. Alur, R., Dill, D.L.: A theory of timed automata. Theor. Comput. Sci. 126(2), 183–235 (1994)
6. Bartocci, E., Bortolussi, L. (eds.): Proceedings First International Workshop on Hybrid Systems and Biology, HSB 2012, Newcastle Upon Tyne, UK, September 3. EPTCS, vol. 92 (2012)
7. Behrmann, G., Fehnker, A., Hune, T., Larsen, K.G., Pettersson, P., Romijn, J., Vaandrager, F.W.: Minimum-cost reachability for priced timed automata. In: Di Benedetto, M.D., Sangiovanni-Vincentelli, A.L. (eds.) HSCC 2001. LNCS, vol. 2034, pp. 147–161. Springer, Heidelberg (2001)
8. Blom, H., Lygeros, J.: Stochastic Hybrid Systems: Theory and Safety Critical Applications, vol. 337 (2006)
9. Bouyer, P., Brihaye, T., Bruyère, V., Raskin, J.F.: On the optimal reachability problem of weighted timed automata. Formal Methods in System Design 31(2), 135–175 (2007)
10. Bouyer, P., Brinksma, E., Larsen, K.G.: Optimal infinite scheduling for multi-priced timed automata. Formal Methods in System Design 32(1), 3–23 (2008)

11. David, A., Du, D., Larsen, K.G., Legay, A., Mikucionis, M., Poulsen, D.B., Sedwards, S.: Statistical model checking for stochastic hybrid systems. In: Bartocci, Bortolussi (eds.) [6], pp. 122–136
12. David, A., Du, D., Larsen, K.G., Mikucionis, M., Skou, A.: An evaluation framework for energy aware buildings using statistical model checking. Science China Information Sciences 55(12), 2694–2707 (2012)
13. David, A., Larsen, K.G., Legay, A., Mikučionis, M., Wang, Z.: Time for statistical model checking of real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 349–355. Springer, Heidelberg (2011)
14. David, A., Larsen, K.G., Legay, A., Mikučionis, M., Poulsen, D.B., van Vliet, J., Wang, Z.: Statistical model checking for networks of priced timed automata. In: Fahrenberg, U., Tripakis, S. (eds.) FORMATS 2011. LNCS, vol. 6919, pp. 80–96. Springer, Heidelberg (2011)
15. David, A., Larsen, K.G., Legay, A., Mikučionis, M., Wang, Z.: Time for statistical model checking of real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 349–355. Springer, Heidelberg (2011), `http://dl.acm.org/citation.cfm?id=2032305.2032332`
16. Hahn, E., Hartmanns, A., Hermanns, H., Katoen, J.P.: A compositional modelling and analysis framework for stochastic hybrid systems. Formal Methods in System Design 43(2), 191–232 (2013), `http://dx.doi.org/10.1007/s10703-012-0167-z`
17. Henzinger, T.A.: The theory of hybrid automata. In: Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science, LICS 1996, pp. 278–292. IEEE Computer Society, Washington, DC (1996)
18. Jongerden, M.R., Haverkort, B.R.: Which battery model to use? IET Software 3(6), 445–457 (2009)
19. Jongerden, M.R., Haverkort, B.R., Bohnenkamp, H.C., Katoen, J.P.: Maximizing system lifetime by battery scheduling. In: DSN. IEEE (2009)
20. Larsen, K.G.: Statistical model checking, refinement checking, optimization, ... for stochastic hybrid systems. In: Jurdziński, M., Ničković, D. (eds.) FORMATS 2012. LNCS, vol. 7595, pp. 7–10. Springer, Heidelberg (2012)
21. Manwell, J.F., McGowan, J.G.: Lead acid battery storage model for hybrid energy systems. Solar Energy 50(5), 399–405 (1993)
22. Martins, J., Platzer, A., Leite, J.: Statistical model checking for distributed probabilistic-control hybrid automata with smart grid applications. In: Qin, S., Qiu, Z. (eds.) ICFEM 2011. LNCS, vol. 6991, pp. 131–146. Springer, Heidelberg (2011)
23. Mikučionis, M., Larsen, K.G., Rasmussen, J.I., Nielsen, B., Skou, A., Palm, S.U., Pedersen, J.S., Hougaard, P.: Schedulability analysis using uppaal: Herschel-planck case study. In: Margaria, T., Steffen, B. (eds.) ISoLA 2010, Part II. LNCS, vol. 6416, pp. 175–190. Springer, Heidelberg (2010)
24. Pulungan, R., Hermanns, H.: Effective minimization of acyclic phase-type representations. In: Al-Begain, K., Heindl, A., Telek, M. (eds.) ASMTA 2008. LNCS, vol. 5055, pp. 128–143. Springer, Heidelberg (2008)
25. Zhang, L., She, Z., Ratschan, S., Hermanns, H., Hahn, E.M.: Safety verification for probabilistic hybrid systems. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 196–211. Springer, Heidelberg (2010)
26. Zuliani, P., Platzer, A., Clarke, E.M.: Bayesian statistical model checking with application to Stateflow/Simulink verification. Formal Methods in System Design 43(2), 338–367 (2013)

Using Statistical Model Checking for Measuring Systems[⋆]

Radu Grosu[1], Doron Peled[2], C.R. Ramakrishnan[3], Scott A. Smolka[3], Scott D. Stoller[3], and Junxing Yang[3]

[1] Vienna University of Technology
[2] Department of Computer Science, Bar Ilan University
[3] Department of Computer Science, Stony Brook University

Abstract. State spaces represent the way a system evolves through its different possible executions. Automatic verification techniques are used to check whether the system satisfies certain properties, expressed using automata or logic-based formalisms. This provides a Boolean indication of the system's fitness. It is sometimes desirable to obtain other indications, measuring e.g., duration, energy or probability. Certain measurements are inherently harder than others. This can be explained by appealing to the difference in complexity of checking CTL and LTL properties. While the former can be done in time linear in the size of the property, the latter is PSPACE in the size of the property; hence practical algorithms take exponential time. While the CTL-type of properties measure specifications that are based on adjacency of states (up to a fixpoint calculation), LTL properties have the flavor of expecting some multiple complicated requirements from each execution sequence. In order to quickly measure LTL-style properties from a structure, we use a form of statistical model checking; we exploit the fact that LTL-style properties on a path behave like CTL-style properties on a structure. We then use CTL-based measuring on paths, and generalize the measurement results to the full structure using optimal Monte Carlo estimation techniques. To experimentally validate our framework, we present measurements for a flocking model of bird-like agents.

1 Introduction

Model checking aims to check that a model of a system satisfies a given specification. Recent results [1,7,9] show how to extend model checking into a more general method for measuring quantitative properties of a given system. Measurements can provide information about time, energy, the probability of an event occurrence, etc. In this paper, we explore the use of statistical model checking techniques for measuring quantitative properties of systems. We illustrate the power of these techniques for measuring the aggregate behavior of a flock of bird-like agents.

[⋆] The 2nd author is supported by ISF grant "Efficient Synthesis Method of Control for Concurrent Systems", award 126/12.

Techniques for the verification of quantitative properties have always relied on the ability to measure the quantities of interest: clock values in real-time systems, probabilities in stochastic systems, etc. There have been several efforts to apply similar techniques for measuring more general quantitative properties based on the operations needed to compute the measurements. For instance, [1] addresses the problems associated with measurements involving operations such as limit average, maximum and discounted sum; and [7] provides a logic for expressing a generalized class of quantitative properties.

In [10], we showed a neighborhood-based measurement scheme that is based on CTL-like specifications: creating a set of nodes in a graph corresponding to each state in the state space, and performing a measurement in a distributed manner propagating values from one node to its neighbors. When performing real-valued measurements, that scheme was limited to finite structures (weighted automata). The neighborhood-based measurement has the characteristic of CTL specification in the sense that the measurement is based on values that can be, temporarily or permanently, assigned to states. When the specification is sequence-based (or path-based), as in the logic LTL, or similar specification formalism that deal with quantitative real-time values that depend on the execution path, this technique does not work: the value of a state can depend on some complicated information related to the path through which the state was reached; states with different histories can have different values, and, to make things more complicated, this is affected from the rest of the execution yet to come. The difference is similar to the difference between CTL and LTL model checking, e.g., see [6] and [8]. While in CTL, we can put a temporary Boolean value per state while calculating subformulas, LTL model checking is done by providing some product of the state space with an automaton representing some essential summary of the sequence so far.

Since we are interested here in sequence based measuring, we make the following observation: on a single sequence, the neighborhood-based technique is the same as for a structure. We thus make our measurement sequence by sequence. We are limited in doing so by the fact that there may be infinitely many sequences and also the sequences themselves can be infinite. Hence, we measure finitely many sequences, using statistical model checking Monte Carlo technique. Moreover, we base our measurements on finite prefixes of executions.

While traditional model checking explores the state space exhaustively, statistical model-checking techniques sample from the state space, ensuring that sufficient samples are drawn, in order to verify a given property with a desired confidence level and error margin. Given a stochastic system S, a Boolean (temporal) property ϕ, and a real-valued parameter θ, statistical model checking determines whether $pr(S \models \phi) \geq \theta$ [14,22]. This paper explores the use of Monte Carlo techniques for simultaneously measuring Boolean and quantitative properties, where the quantitative measures are dependent on the Boolean part.

Technical Approach and Contributions

1. *Specification of measurement computations.* Following [10], we describe a specification formalism closely resembling a synchronous dataflow language for measurement computation. At a high level, we associate a set of measurement variables with each state in the state space. We specify a mechanism for computing the values of variables at a state, based on the values of variables at the state's neighbors. The formalism is general enough to encode bounded model checkers (MCs) [13,16] for Boolean temporal properties expressed in CTL or LTL. The advantage of the proposed measurement specification and related MC algorithm is in its simplicity and efficiency. It allows one to assemble the measurement specification from subproperties, just as CTL combines its temporal specification from its subformulas. The synchronous dataflow framework can be viewed as a generalization of *testers* proposed in [18]. Section 3 describes the specification formalism in greater detail.

2. *Model measurement using Monte Carlo techniques.* The computational mechanism described by the synchronous dataflow formalism associates a set of measures with each system execution (which may be trace or tree, depending on whether the property is linear or branching-time). The quantity of interest at a higher level may be an aggregate property covering the set of possible system executions (e.g., the average over possible runs). To this end, we develop a Monte Carlo technique (MCT) for generating a suitable set of samples (traces or trees) so that aggregate quantities can be computed to a desired confidence bound. The novelty of our MCT is that it jointly computes Boolean (satisfaction) and real (e.g., mean) values, using information from both strands to improve efficiency. Section 4 describes our MCT.

3. *An integrated model of flocking.* A number of different flocking models (FMs) have been developed to describe and explain the flocking behavior of birds [19,21,15,4,3]. The FMs generally consider each bird as an agent, where all agents are governed by the same control law. Certain input variables in the law executed by each agent are based on the attributes (e.g., velocity, position) of other agents in the flock. Such FMs are useful in understanding how emergent behaviors of the collection of agents arise from decisions made individually by each agent. We consider an integrated FM that uses a control law comprising a variety of terms from the existing literature. We consider quantitative objectives for the flock's behavior (e.g., velocity matching, the extent to which the velocities of agents are aligned), given in our measurement-specification formalism. The results of the measurement can be used to synthesize parameters (e.g., weights of different terms in the flocking control law) that optimize the objectives. Section 2 describes the FM in detail.

Section 5 reports preliminary results for measuring the velocity-matching objective of the flocking model. The results provide insight into the effectiveness of two approaches to controlling the accuracy of the Monte Carlo estimation.

2 Flocking

We illustrate our method with measurements of the behavior of a flock of agents. The flocking model is biologically inspired and may be useful in the design of controllers for unmanned vehicles. In our flocking model, autonomous agents move in 2-dimensional space; the generalization to 3 dimensions is straightforward. Each agent's motion is determined by a locally executed control law. Every agent runs the same control law. Each agent has sensors that report the positions and velocities of all agents. The control law takes that information as input and returns an acceleration (i.e., change in velocity) for the agent.

Our broader goal is to develop methods for the design of control laws that cause the flock's behavior to satisfy given Boolean and quantitative objectives. An example of a Boolean objective is *collision avoidance*, i.e., that agents always maintain a specified minimum separation from each other. An example of a quantitative objective is *velocity matching* (VM), i.e., that agents gradually match velocities with each other, so the entire flock moves together. Note that this is a quantitative objective when the goal is to maximize VM, and a Boolean objective when the goal is to achieve VM above a specified threshold.

The control law typically contains several terms, each aimed at satisfying one or more objectives, and numerous parameters, including a weight for each term. Our broader goal is to develop methods that find values for the parameters that best achieve the specified Boolean and quantitative objectives. The two key components of the design method are an optimization framework and a measurement framework, used to measure how well the behavior of a flock with a given control law satisfies specified objectives.

There is some existing work on using genetic algorithms to adjust parameters in flocking control laws [2,20]. They consider relatively limited and specific forms for the flocking control law and the objectives; in particular, the objective is expressed as a fitness function that is simply hand coded in a programming language. In contrast, our work aims to be more general and flexible, both in terms of considering a larger variety of terms in the flocking control law, including the terms in the flocking models in [19,21,15,4,3], and considering more varied and complex objectives, expressed more abstractly in a measurement framework.

Running Example. To illustrate the ideas in this paper, we consider a control law with a few selected terms. Let $x(t)$ and $v(t)$ be the vector of 2-dimensional positions and velocities, respectively, of all agents at time t. Let $x_i(t)$ and $v_i(t)$ be the position and velocity, respectively, of agent i at time t. Let k be the number of agents. The equation of motion and the control law are given in Figure 1. The acceleration is a weighted sum of the terms described next, with the *speed limit* function *spdLim* (formal definition omitted) applied to the sum to ensure that the magnitude of the velocity does not exceed $v_{\max} = 2$.

The *velocity-averaging* term *va*, adopted from Cucker and Dong's model [3], is designed to align the velocities of all agents, by gradually shifting them towards the flock's average velocity. The *strength function* ϕ specifies the strength of the velocity-matching influence between two agents as a function of the distance between them; H, β, ..., are parameters of the model.

$$\begin{aligned}
\dot{x}_i(t) &= v_i(t) && (1)\\
\dot{v}_i(t) &= \mathit{spdLim}(w_{va}\cdot va_i(t) + w_{ca}\cdot ca_i(t) + w_{ctr}\cdot ctr_i(t) + w_{rpl}\cdot rpl_i(t)) && (2)\\
va_i(t) &= \sum_{j=1}^{k} \phi(||x_i(t)-x_j(t)||)(v_j(t)-v_i(t)) && (3)\\
\phi(r) &= \frac{H}{(1+r^2)^\beta} && (4)\\
ca_i(t) &= \mathit{VM}(t)\sum_{j\neq i} f(||x_i(t)-x_j(t)||^2)(x_i(t)-x_j(t)) && (5)\\
\mathit{VM}(t) &= (\frac{1}{k}\sum_{i>j}||v_i(t)-v_j(t)||^2)^{\frac{1}{2}} && (6)\\
f(r) &= (r-d_{ca})^{-2} && (7)\\
ctr_i(t) &= \frac{||v_i(t)||}{||\mathit{relCtr}_i(t)||}\cdot \mathit{relCtr}_i(t) - v_i(t) && (8)\\
\mathit{nbors}_i(t,d) &= \{j \mid j\neq i \wedge ||x_j(t)-x_i(t)|| \leq d\} && (9)\\
\mathit{relCtr}_i(t) &= \left(\frac{1}{|\mathit{nbors}_i(t)|}\sum_{j\in \mathit{nbors}_i(t,d_{ctr})} x_j(t)\right) - x_i(t) && (10)\\
rpl_i(t) &= \frac{||v_i||}{||\mathit{offset}_i(t)||}\cdot \mathit{offset}_i(t) - v_i && (11)\\
\mathit{offset}_i(t) &= \frac{1}{|\mathit{nbors}_i(t,d_{rpl})|}\sum_{j\in \mathit{nbors}_i(t,d_{rpl})}(x_j - x_i) && (12)
\end{aligned}$$

Fig. 1. Flocking model

The *collision avoidance* term ca, adopted from Cucker and Dong's model [3], is designed such that the separation between every pair of agents is always larger than d_{ca}^2. The *velocity matching* function VM (called *alignment measure* in [3]) measures the alignment the velocities of all agents (smaller values indicate better alignment). The *repelling function* f specifies the strength of the collision-avoidance influence between two agents as a function of the distance between them.

The *centering* term ctr, adopted from Reynolds' model [19], causes agents to form cohesive groups (sub-flocks), by shifting each agent's velocity to point towards the centroid (i.e., average) of the positions of its d_{ctr}-neighbors, where an agent's *d-neighbors* are the agents within distance d of the agent. The function $\mathit{nbors}_i(t,d)$ returns the set of indices of d-neighbors of agent i at time t. The function $\mathit{relCtr}_i(t)$ returns the relative position (i.e., relative to agent i's current position) of the centroid of its d_{ctr}-neighbors at time t. The definition of $ctr_i(t)$ applies a scaling factor to $\mathit{relCtr}_i(t)$ that yields a vector with the same magnitude as $v_i(t)$ and pointing in the same direction as $\mathit{relCtr}_i(t)$.

The *repulsion term*, adopted from Reynolds' model [19], causes agents to move away from their d_{rpl}-neighbors. The function $\mathit{offset}_i(t)$ returns the average of the offsets (i.e., differences in position) between agent i and its d_{rpl}-neighbors. The scaling factor applied to the offset in the definition of $rpl_i(t)$ is similar to the scaling factor used in the centering term.

The initial state is chosen stochastically. In our experiments, agents' initial positions are chosen uniformly at random in the box $[0..k, 0..k]$, and their initial velocities are chosen uniformly at random in $[0..1, 0..1]$. In more detailed models, stochastic environmental factors (e.g., wind) can be modeled with probabilistic transitions. In this case, the simulator would select from the corresponding probability distribution when the transition is taken.

Our experiments use the following parameter values: $k = 10, w_{va} = 0.6, w_{ca} = 0.1, w_{ctr} = 0.2, w_{rpl} = 0.1, H = 0.1, \beta = 1/3, d_{ca}^2 = 0.1, d_{ctr} = 5, d_{rpl} = 3$. We simulate the behavior of the flock using discrete-time simulations with a time step of 1 second and a duration of 50 seconds of simulation time. We chose this duration since we experimentally observed that the velocity-matching objective function stabilizes within 50 steps (a larger value had little effect).

Objective: Velocity Matching. We consider velocity matching as an objective for the running example. We use the velocity-matching function VM in equation (6) to measure how well the velocities are aligned in a state. Note that smaller values indicate better alignment. We consider two aspects of how well velocity matching is achieved: (1) how long it takes for VM to fall below a specified threshold θ and remain below θ for the rest of the execution; we measure this time as a fraction of the duration of the simulated execution, so the value is between 0 and 1; and (2) the average value of VM after it falls below the threshold θ and remains there; we measure this as a fraction of the maximum possible value of VM, so the value is also between 0 and 1. To combine these two quantities into a single fitness measure suitable for use in an optimization framework, we take a linear combination of them, with weights 0.01 and 0.99, respectively, so that these two quantities are of the same scale. In Section 3, we describe how to formally compute this linear combination using an LTL-style measurement.

3 Neighborhood-Based and Sequence-Based Measurements

In this section, we provide a formalism for expressing neighborhood-based measurements. Our formalism is state-based: each state is assumed to contain a tuple of constants and measurement variables. The variables are initiated, then, at each clock tick, the states, synchronously, exchange their values with their neighbor states, and apply an update function to obtain a new measurement. An expression over the state variables is assigned to each state, and its value is calculated at each tick. The value of the expression must decrease with each update, so that it bounds the amount of steps that can be performed by each state.

Definition 1. *A* state space $\mathcal{S}$ *is a triple* $\langle S, s_0, R\rangle$ *where*

S is a finite set of states.
$s_0 \in S$ is the initial state*.*
$R \subseteq S \times S$ is a relation over S, where if $(s, s') \in R$, then s' is the successor *of s.*

Let $R^\bullet(s) = \{q|R(s,q)\}$ be the successors *of s, ${}^\bullet R(s) = \{q|R(q,s)\}$ its predecessors, and $N(s) = {}^\bullet R(s) \cup R^\bullet(s)$. Let $n(s) = |N(s)|$ be the number of neighbors of S. We assume that the size z of the state space, and the length of its maximal path (its* width*) w are known.*

Often, the state space is generated from some given system, where the states represent, e.g., the assignment of values to variables, and a successor state is generated from its predecessor by firing some atomic transition. The connection between a system and its state space is orthogonal to the focus of this paper.

3.1 Neighborhood-Based Measurement

In [10], we propose a measuring specification based on neighborhoods for a state space. We associate with each state the following components:

- A tuple of *measurement* variables V over some finite domains.
- Initial value for each measurement variable from its domain.
- A well founded set $\langle W, <\rangle$, where W is a set of values and $<$ is a partial order on W where no infinite decreasing chain exists.
- An expression E, based on the variables V that results in values from W. We denote the current value of E at state s as $E(s)$.
- Each state may have some constants assigned to it (this can be extended so that constants are assigned also to edges).
- An *update* function f for the variables V. It can be based on the current version T of the variables (denoted by $T(s)$), the constants on the state, and a version V^q for each neighbor q of s (also, constants on the edges to and from neighbors). For some purposes, it is sufficient to use updates based only on successors or on predecessors. The update function must satisfy that $E > E'$, where E' is the expression E after applying the update.

The measuring specification is, in itself, also an algorithm, which can be implemented directly. Basically, it consists of the following:

```
With each tick of the clock, execute per each state of the state $s$ space:
    If $E(s)$ is not minimal, then do
      Send $T(s)$ to all neighbors.
      Receive $T^q(s)$ from the neighbors, $q \in \{q_1, q_2, \ldots q_{n(s)}\}$ of $s$.
      Let $V := f(V, V^{q_1}, \ldots, V^{q_{n(s)}})$ od
```

Note that with the recalculation of the values of V in the state, the expression E would decrease.

Example. An example of measuring is to find what is the maximal value assigned to any node reachable from the current node. We set up the following:

- A constant c per each state (denoted $c(s)$) representing the measured value.
- A variable m that contains the current calculated maximum, initialized to c.
- A counter d initialized to w, the width of the structure.
- The well founded domain is the natural numbers with the usual $<$. The decreasing expression E assigned to each state is simply d.
- The update function takes the values of successor states $R^\bullet$ and calculates the multiple assignment

$$(m, d) := (max(m, m^1, m^2, \ldots m^{n(s)}), d - 1)$$

We show now (see also [10]) an implementation of CTL model checking using our measuring principles. This algorithm resembles the original Clarke and Emerson algorithm [6], with the help of bounded model checking. This is just an example to show that the power of our measuring formalism exceeds that of CTL model checking. However, one may want to use the formalism without fixing a different temporal property as the basis for measurements.

Recall that the syntax of the CTL formulas is as follows:

$$\varphi := p|\neg\varphi|(\varphi \vee \varphi)|EX\varphi|(\varphi \wedge \varphi)|(\varphi EU\varphi)|(\varphi AU\varphi)$$

The semantics is as usual, see [6].

First, the variables for model checking a CTL property φ include for each subformula η a variable v_η. These variables have three possible values: **U**, **T** and **F**, where **U** (for *undefined*) being the initial value. There is also a *phase counter* variable *pc*, which is set initially to the number of subformulas in φ, and a *downcounter dc*. The downcounter is set to 1 at the beginning phases that are associated with Boolean subformulas or EX, and to w (or to z) when the phase is associated with *EU* or *AU*. This is because information as far as s state away from the current state may affect the value of the *EU* or *AU* formula.

Now, at each step, we decrement one from *dc*, and if it reaches 0, decrement one from *pc*, unless it is zero, and then we terminated. When we move to a new phase, we set *dc* to 1 or w, according to the type of the subformula. We handle the subformulas according to their size. In this way, when dealing with some subformula η, the measurement variables for its subformulas are already calculated and set to **T** or **F**. Depending on the type of the subformula, we perform an action. For a Boolean operators, we perform the same operator on the values of the corresponding measurement variables, e.g., if the subformula is $(\eta \vee \rho)$ then we set $v_{(\eta\vee\rho)}$ to $v_\eta \vee v_\rho$. Since we use three valued logic, we need to extend the Boolean operators. Accordingly, we have $(\mathbf{T} \vee \mathbf{U}) = \mathbf{T}$, $(\mathbf{F} \wedge \mathbf{U}) = \mathbf{F}$, and the symmetric situations. For the other cases involving at least one **U**, the result is **U**.

For the subformula $(\eta EU\rho)$, we obtain at each step the values of $v^{q_i}_{(\eta EU\rho)}$ from each successor $q \in N(s)$. The values v_ρ and v_η are calculated in a previous phase. Then we set up

$$v_{(\eta EU\rho)} := v_\rho \vee (v_\eta \wedge \bigvee_{q\in N(s)} v^q_{(\eta EU\rho)})$$

For $(\eta A U \rho)$, just replace in the formula above $\bigvee_{i \in 1..m}$ with $\bigwedge_{i \in 1..m}$. Now, if we finished the current phase (dc becomes 0) and $v_{(\eta E U \rho)}$ is still $\mathbf{U}$, then we set it to $\mathbf{F}$.

In [10], we showed how to use such a measuring specification for calculating whether a robot can be reached into some docking station before its battery is critically discharged. There, we used the backwards propagation of values to check whether the shortest paths from each state are still short enough we used forwards propagation to check whether critical discharge (including over cycles in the path of the robot) occur.

3.2 Path Measurements

We discuss now measurements that depends on the history of the execution, as well as its future. While our measurement formalism does not depend on a logic, it is easy to explain the different characterization of such measurement using the different requirements between verifying the temporal logics CTL and LTL. The temporal logic LTL [17] has a different characterization than CTL, since it can claim multiple properties for each single execution path. In fact, there is an exponential number of ways, in the size of the LTL property, in which a state can evolve to satisfy the property, depending on its history. Model checking of LTL properties is facilitated by a product of the state space with an automaton that represents updating and memorizing the available possibilities. For this reason, the neighborhood based measurement we proposed in [10] would make very little sense for path based properties (such as LTL formulas).

There are two problems that we need to face in such measuring:

1. The paths that are measured may be infinite. Although it is known that if there is an execution that satisfies an LTL property then there is one that is ultimately periodic (see, e.g., [8]), i.e., consists of a finite prefix, and a finite part that repeats indefinitely. However, measuring non-Boolean properties may have different results where ultimately periodic sequences are not good representatives (see, e.g., [1] for measuring the limit of the average of values along a sequence).
2. There are multiple paths in the structure (possibly infinite), and we are interested to give a measurement of all the paths, or sufficiently many of them.

In order to tackle problem 1, we assume measurements that are affected mainly by a finite prefix of sequences. We may then decide to use some limit on the length of a sequence, and show, separately, how measurements are affected by changing this limit. In order to tackle problem 2, we use generalized Monte-Carlo measurements, and are satisfied when we can conclude that a large enough number of executions (defined as a parameter), has guaranteed some measurement threshold. This means that we have to provide the threshold (some value that the measurements either surpass or do not reach), and a the level of confidence required for this threshold.

3.3 Example: Velocity Matching Based Measurement

As an example, we show how to measure the objective of velocity matching in Section 2 using our measurement formalism. Although the measurement does not necessarily depend on having a temporal property associated with the measure (e.g., calculating the average, the sum, etc.) we can, in this case look at the LTL property $\varphi_{VM} := FGp$, (eventually always p), where p is the proposition $VM \leq \theta$, and VM is the (normalized) velocity matching in a state.

Note that since in path measurement we have only a single successor and a single predecessor, we can distinguish them, e.g., denote the successor version of v by v' (and if we look at the predecessors, denote the predecessor of v by v'').

The variables we use are as follows:

- Boolean variables: B_p, B_{Gp} and B_{FGp}, all initialized to $\mathbf{U}$.
- Real variables:
  - vm: the velocity matching value, initialized to 0.
  - avg: the average velocity matching when Gp holds, initialized to 0.
  - $step$: the number of steps from current state to the first state where Gp holds, initialized to ∞.
  - lc: the linear combination of avg and $step$, initialized to 0.
- Down counter: l, which is initialized to w, the length of the paths we use (we use a constant length, which is 50)
- The values got from the successor are marked by a prime, i.e., ${B_{Gp}}'$, ${B_{FGp}}'$, avg', etc. For the tester corresponding to the last state of the path who has no successor, we assume the following: ${B_{Gp}}' = \mathbf{T}$, ${B_{FGp}}' = \mathbf{F}$, $avg' = 0$, $step' = \infty$.

At each step, we calculate the following:

- If $l > 0$:
  - $B_p = p$.
  - $B_{Gp} = B_p \wedge {B_{Gp}}'$.
  - $B_{FGp} = B_{Gp} \vee {B_{FGp}}'$.
  - $vm = VM(s)$, the velocity matching value in current state s. The calculation of $VM(s)$ is explained in Section 2.
  - $avg = \textbf{if } (B_{Gp} = \mathbf{T}) \textbf{ then } (avg' * (w - l) + vm)/(w - l + 1) \textbf{ else } avg'$.
  - $step = \textbf{if } (B_{Gp} = \mathbf{T}) \textbf{ then } 1 \textbf{ else } (step' + 1)$.
  - $lc = 0.99 * avg + 0.01 * (step/w)$.
  - $l := l - 1$.

The well founded ordering is the value of l. That is, we terminate when $l = 0$. At this point, we can take values (B_{FGp}, lc) from the initial node of the sequence.

4 Generalized Monte-Carlo Measurements

The formalism discussed in Section 3 takes as input a bounded sequence $s_{1:n}$ and returns a measure of it. For example, for the flocking model (FM) we presented in Section 2, we return a pair (b, r) of a Boolean and real value, respectively, where

b is the value of $FG(VM \leq \theta)$, and r is the weighted sum of two quantities: how long it takes (as a fraction of n) for VM to fall below threshold θ and remain there; and, when b is true, the average of VM for the (maximal) subsequence where $G(VM \leq \theta)$ holds.

A particular sequence $s_{1:n}$ is generated by running a k-agent FM for a given initial state, which is assumed to be distributed in the box $[0..k, 0..k]$, for agent positions, and in the box $[0..1, 0..1]$, for agent velocities. The FM assumes that these distributions are uniform, but, in general, they can be any arbitrary distribution, including Gaussian.

Since each pair (b, r) is the result of an initialized execution of the FM, both b and r are the values of *independent, identically distributed* (IID) random variables $Z = (B, R)$. Assuming that r belongs to the interval $[0, 1]$ is not a limitation, as one can always normalize the values of R, provided that one knows its range. We do exactly this in Sections 2-3, where the fitness value of the flock we compute is the weighted sum of two $[0, 1]$-normalized quantities.

A generalized measurement aims to efficiently obtain a joint estimate $\mu_Z = (\mu_B, \mu_R)$ of the mean values $\mathbf{E}[B]$, $\mathbf{E}[R]$ of B and R, respectively. Since an exact computation of μ_Z is almost always intractable (e.g. NP-hard), a Monte Carlo (MC) approach is used to compute an (ϵ, δ)-approximation of this quantity.

The main idea of MC is to use N random variables (RVs) $Z_1, \ldots, Z_N$, also called *samples*, IID distributed according to Z and with mean μ_Z, and to take the sum $\widetilde{\mu}_Z = (Z_1 + \ldots + Z_N)/N$ as the value approximating the mean μ_Z.

While MC techniques were used before to measure the satisfaction probability of temporal logic formulas [11,9,14], or to compute the mean of an RV [5,12], the main novelty of this paper is to *jointly* measure the Boolean satisfaction and the mean real value. The Boolean-value computation is informing the real-value computation and *vice versa*, thereby increasing the efficiency of our approach.

Additive approximation. An important issue in an MC approximation scheme is to determine an appropriate value for N. If μ_Z is expected to be large, then one can exploit the Bernstein inequality and fix N to be $\Upsilon \propto ln(1/\delta)/\epsilon^2$. This results in an *additive* or *absolute-error* (ϵ, δ)-*approximation scheme*:

$$\mathbf{Pr}[\mu_Z - \epsilon \leq \widetilde{\mu}_Z \leq \mu_Z + \epsilon] \geq 1 - \delta$$

where $\widetilde{\mu}_Z$ approximates μ_Z with absolute error ϵ and with probability $1 - \delta$.

If Z is assumed to be a Bernoulli RV, then one can use the Chernoff-Hoeffding instantiation of the Bernstein inequality, and further fix the proportionality constant to $\Upsilon = 4\, ln(2/\delta)/\epsilon^2$, as in [11]. This bound can also be used for the joint estimation of RV $Z = (B, R)$, as B is a Bernoulli RV, and the proportionality constraint of the Bernstein inequality is also satisfied for RV R. This results in the *additive approximation algorithm* (AAA) below.
It is important to note that in AAA, the number of samples N depends only on ϵ and δ, and is totally oblivious to the mean value μ_Z to be computed.

Multiplicative approximation. In case the mean value μ_Z is very low, the additive approximation $\widetilde{\mu}_Z$ may be meaningless, as the absolute error may be considerably

AAA algorithm
input: (ϵ, δ) with $0 < \epsilon < 1$ and $\delta > 0$.
input: Random vars Z_i with $i > 0$, IID.
output: $\widetilde{\mu}_Z$ approximation of μ_Z.

```
(1)  $\Upsilon = 4\ln(2/\delta)/\epsilon^2$;
(2)  for (N=0; N$\leq\Upsilon$; N++) S=S+$Z_N$;
(3)  $\widetilde{\mu}_Z$=S/N; return $\widetilde{\mu}_Z$;
```

larger than the actual value μ_Z. In such cases, a *multiplicative* or *relative-error* (ϵ, δ)*-approximation scheme* is more appropriate:

$$\mathbf{Pr}[\mu_Z - \mu_Z\epsilon \leq \widetilde{\mu}_Z \leq \mu_Z + \mu_Z\epsilon] \geq 1 - \delta$$

where $\widetilde{\mu}_Z$ approximates μ_Z with relative error $\mu_Z\epsilon$ and with probability $1 - \delta$.

In contrast to the Chernoff-Hoeffding bound $\Upsilon = 4\ln(2/\delta)/\epsilon^2$, required to guarantee an absolute error ϵ with probability $1 - \delta$, the *zero-one estimator theorem* in [12] requires a bound proportional to $N = 4\ln(2/\delta)/\mu_Z\epsilon^2$ to guarantee a relative error $\mu_Z\epsilon$ with probability $1 - \delta$.

When applying the zero-one estimator theorem, one encounters, however, two main difficulties. The first is that N depends on $1/\mu_Z$, the inverse of the value that one intends to approximate. This problem can be circumvented by finding an upper bound κ of $1/\mu_Z$ and using κ to compute N. Finding a tight upper bound is, however, in most cases very difficult, and a poor choice of κ leads to a prohibitively large value for N.

An ingenious way of computing N without relying on μ_Z or κ is provided by the *Stopping Rule Algorithm* (SRA) of [5]. When $\mathbf{E}[Z] = \mu_Z > 0$ and $\Sigma_{i=1} Z_i \geq \Upsilon$, the expectation $\mathbf{E}[N]$ of N equals Υ.

SRA algorithm
input: (ϵ, δ) with $0 < \epsilon < 1$ and $\delta > 0$.
input: Random vars Z_i with $i > 0$, IID.
output: $\widetilde{\mu}_Z$ approximation of μ_Z.

```
(1)  $\Upsilon = 4(e-2)\ln(2/\delta)/\epsilon^2$; $\Upsilon_1 = 1 + (1+\epsilon)\Upsilon$;
(2)  for (N=0, S=0; S$\leq\Upsilon_1$; N++) S=S+$Z_N$;
(3)  $\widetilde{\mu}_Z$=S/N; return $\widetilde{\mu}_Z$;
```

The second difficulty in applying the zero-one estimator theorem is the factor $1/\mu_Z\epsilon^2$ in the expression for N, which can render the value of N unnecessarily large. A more practical approach is offered by the *generalized zero-one estimator theorem* of [5] which states that N is proportional to $\Upsilon' = 4\rho_Z \ln(2/\delta)/(\mu_Z\epsilon)^2$, where $\rho_Z = \max\{\sigma_Z^2, \epsilon\mu_Z\}$ and σ_Z^2 is the variance of Z. Thus, if σ_Z^2, which equals $\mu_Z(1 - \mu_Z)$ for Z a Bernoulli random variable, is greater than $\epsilon\mu_Z$, then $\sigma_Z^2 \approx \mu_Z$, $\rho_Z \approx \mu_Z$ and therefore $\Upsilon' \approx \Upsilon$. If, however, σ_Z^2 is smaller than $\epsilon\mu_Z$, then $\rho_Z = \epsilon\mu_Z$ and Υ' is smaller than the Υ by a factor of $1/\epsilon$.

To obtain an appropriate bound in either case, [5,9] have proposed the *optimal approximation algorithm* (OAA) shown above. This algorithm makes use of the

OAA algorithm

```
input:   Error margin $\epsilon$ and confidence ratio $\delta$ with $0 < \epsilon \leq 1$ and $0 < \delta \leq 1$.
input:   Random vars $Z_i, Z'_i$ with $i > 0$, IID.
output:  $\widetilde{\mu}_Z$ approximation of $\mu_Z$.

(1)  $\Upsilon = 4\,(e - 2)\,\ln(2/\delta)/\epsilon^2$; $\Upsilon_2 = 2\,(1 + \sqrt{\epsilon})\,(1 + 2\sqrt{\epsilon})\,(1 + \ln(3/2)/\ln(2/\delta))\,\Upsilon$;
(2)  $\widehat{\mu}_Z$ = SRA(min{1/2, $\sqrt{\epsilon}$}, $\delta$/3, Z);
(3)  N = $\Upsilon_2\,\epsilon / \widehat{\mu}_Z$; S = 0;
(4)  for (i=1; i$\leq$N; i++) S = S + $(Z'_{2i-1} - Z'_{2i})^2/2$;
(5)  $\widehat{\rho}_Z$ = max{S/N, $\epsilon\,\widehat{\mu}_Z$};
(6)  N = $\Upsilon_2\,\widehat{\rho}_Z / \widehat{\mu}_Z^2$; S = 0;
(7)  for (i=1; i$\leq$N; i++) S = S + $Z_i$;
(8)  $\widetilde{\mu}_Z$ = S / N; return $\widetilde{\mu}_Z$;
```

outcomes of previous experiments to compute N, a technique also known as *sequential MC*. The OAA algorithm consists of three steps. The first step calls the SRA algorithm with parameters $(\sqrt{\epsilon}, \delta/3)$ to obtain an estimate $\widehat{\mu}_Z$ of μ_Z. The choice of parameters is based on the assumption that $\rho_Z = \epsilon\mu_Z$, and ensures that SRA takes $3/\epsilon$ less samples than would otherwise be the case. The second step uses $\widehat{\mu}_Z$ to obtain an estimate of $\widehat{\rho}_Z$. The third step uses $\widehat{\rho}_Z$ to obtain the desired value $\widetilde{\mu}_Z$. Should the assumption $\rho_Z = \epsilon\mu_Z$ fail to hold, the second and third steps will compensate by taking an appropriate number of additional samples. As shown in [5], OAA runs in an expected number of experiments that is within a constant factor of the minimum expected number.

For simplicity, both in SRA and in OAA, we only showed the joint variable $Z = (B, R)$, and used a generic RV $\widehat{\rho}_Z$. In our implementation, however, we distinguish between the mean and the variances of B and R, and if we observe that the variance of R is very low, we stop, even if the variance of B is larger, as R is determining the value of B.

5 Experimental Results

We have implemented our flocking model presented in Section 2 in MATLAB. Recall that the property we measure is φ_{VM} defined in Section 3.

In order to get an *approximate measure for the flocking model*, within a required error margin ϵ and confidence level δ, we applied the generalized Monte-Carlo estimation (GMCE) algorithms discussed in Section 4. The GMCE algorithms use both the the boolean values b and the real values r obtained from the *path measurements* (b, r), in order to determine the stopping time, and to compute the mean values μ_B and μ_R of interest.

Every path measurement (b, r) is obtained by running the measurement algorithm defined in Section 3.3, over a *random execution of the flocking model*. This execution is generated by our *discrete-time simulator*, by first choosing the initial state uniformly at random and then integrating the differential equations given in Section 2, Figure 1.

Table 1 shows the results of 5 runs of OAA with $\theta = 0.05$, $\epsilon = 0.3$ and $\delta = 0.3$. In the table, μ_R is the real part of the output of OAA; it estimates the mean of the quantitative measurement we defined in Section 3.3 for φ_{VM}. N is the number of samples used to compute μ_R.

We omit the boolean part μ_B because it is *always one* for this example. We also show the average (Avg) and the standard deviation (Std) of the results. We compare our results with the AAA algorithm where a fixed number of samples N is used, $N = 4log(2/\delta)/\epsilon^2$, as shown in Table 2. The OAA algorithm requires significantly more samples, but considerably reduces the standard deviation.

Table 1. Results obtained from OAA

Runs	μ_R	N
1	0.00644	66846
2	0.00651	65592
3	0.00646	66696
4	0.00648	66173
5	0.00646	66588
Avg	0.00647	66379
Std	2.659e-05	505.9

Table 2. Results obtained from AAA

Runs	μ_R	N
1	0.00681	84
2	0.00685	84
3	0.00589	84
4	0.00630	84
5	0.00585	84
Avg	0.00634	84
Std	0.00047	0

Note that, in Table 2, we use the same error margin and confidence level as in Table 1. This additive approximation is, however, meaningless, because the additive error is much greater than μ_R itself. If we want to achieve the same accuracy as OAA, the error margin for AAA should be set to $\epsilon' = \mu_R * \epsilon \approx 0.00194$. This results in the sample size $N = 4log(2/\delta)/\epsilon'^2 = 2,016,282$, which is significantly larger than the sample size used in OAA.

Table 3 shows the results when we choose different values of ϵ and δ for the OAA algorithm. It is clear that choosing smaller values of ϵ and δ results in smaller standard deviations at the expense of a larger sample size. We obtain values of μ_R that are, however, highly consistent with one another from different values of ϵ and δ.

Table 3. OAA with different ϵ and δ

	$\epsilon = 0.1$, $\delta = 0.3$	$\epsilon = 0.1$, $\delta = 0.5$	$\epsilon = 0.3$, $\delta = 0.3$	$\epsilon = 0.3$, $\delta = 0.5$
Avg μ_R	0.00647	0.00648	0.00646	0.00648
Std μ_R	1.816e-05	2.370e-05	2.659e-05	3.267e-05
Avg N	131861	102412	66379	51410
Std N	326.2	447.7	505.9	216.8

6 Conclusions

Algorithmic methods for checking the consistency between a system and its specification [6] have been generalized into measuring properties of systems [1].

The study of such measurement techniques provided interesting algorithmic, complexity and computability results.

In a previous work [10] we proposed a simple formalism for fast measurement, based on repeatedly observing the neighborhood of the states of the system. Information about partial measurements flow through the states to and from its neighbor states. Instead of providing a global logic base specification, which assert about the different paths of the state space, and their interconnection, our formalism is based on the view of a state and the information that flows through it from its predecessors and successors. An expression over some well founded set is used to control the termination of the accumulative data flow based measuring. This setting is quite general, and allows measurements that combine different arguments, both Boolean and numeric. The formalism is simple, and the measurement is efficient. If the measurement is provided in terms of some logic formalism, then it needs to be translated into our formalism.

Our measurement formalism is not appropriate for any formalism. An important class of such formalisms are those that are dependent on some memory accumulated on the execution path. In such specification, different paths that lead to the same state may result in different measurements. Even if the amount of memory needed to keep track of the history of the path is finite, the measurement is not unique per given state (as it depends on the history of the path). Performing the measurement path by path is often not feasible as there can be infinitely many or prohibitively many paths. We propose here a tradeoff between accuracy and exhaustiveness, based on statistical model checking [9,11,12,20]. We neighborhood measurement techniques to a statistically big enough sample of paths, and use statistical inference to conclude the measurement of the system.

Such complex measurements can appear naturally in systems that combine physical parameters and in biological systems. As a running example we used models for birds flocking [19,21,15,4,3,2,20]. Based on several researched models for flocking, we want to measure the well behavior of their combination. In particular, the eventual well matching of speeds among birds. We cast the eventual speed matching measurement in terms of our formalism. Then we make experiments based on statistical model checking implemented using MATLAB.

References

1. Chatterjee, K., Doyen, L., Henzinger, T.A.: Expressiveness and closure properties for quantitative languages. Logical Methods in Computer Science 6(3) (2010)
2. Conley, J.F.: Evolving boids: Using a genetic algorithm to develop boid behaviors. In: Proceedings of the 8th International Conference on GeoComputation (GeoComputation 2005) (2005), `http://www.geocomputation.org/2005/`
3. Cucker, F., Dong, J.G.: A general collision-avoiding flocking framework. IEEE Trans. on Automatic Control 56(5), 1124–1129 (2011)
4. Cucker, F., Smale, S.: Emergent behavior in flocks. IEEE Trans. on Automatic Control 52(5), 852–862 (2007)
5. Dagum, P., Karp, R., Luby, M., Ross, S.: An optimal algorithm for Monte Carlo estimation. SIAM Journal on Computing 29(5), 1484–1496 (2000)

6. Emerson, E.A., Clarke, E.M.: Characterizing correctness properties of parallel programs using fixpoints. In: de Bakker, J.W., van Leeuwen, J. (eds.) ICALP 1980. LNCS, vol. 85, pp. 169–181. Springer, Heidelberg (1980)
7. Finkbeiner, B., Sankaranarayanan, S., Sipma, H.: Collecting statistics over runtime executions. Formal Methods in System Design 27(3), 253–274 (2005)
8. Gerth, R., Peled, D., Vardi, M.Y., Wolper, P.: Simple on-the-fly automatic verification of linear temporal logic. In: PSTV, pp. 3–18 (1995)
9. Grosu, R., Smolka, S.: Quantitative model checking. In: Proc. of the 1st International Symposium on Leveraging Applications of Formal Methods (ISOLA 2004), Paphos, Cyprus, pp. 165–174 (November 2004)
10. Grosu, R., Peled, D., Ramakrishnan, C.R., Smolka, S.A., Stoller, S.D., Yang, J.: Compositional branching-time measurements. In: Bensalem, S., Lakhneck, Y., Legay, A. (eds.) From Programs to Systems. LNCS, vol. 8415, pp. 118–128. Springer, Heidelberg (2014)
11. Hérault, T., Lassaigne, R., Magniette, F., Peyronnet, S.: Approximate probabilistic model checking. In: Steffen, B., Levi, G. (eds.) VMCAI 2004. LNCS, vol. 2937, pp. 73–84. Springer, Heidelberg (2004)
12. Karp, R., Luby, M., Madras, N.: Monte-Carlo approximation algorithms for enumeration problems. Journal of Algorithms 10, 429–448 (1989)
13. Latvala, T., Biere, A., Heljanko, K., Junttila, T.A.: Simple bounded LTL model checking. In: Hu, A.J., Martin, A.K. (eds.) FMCAD 2004. LNCS, vol. 3312, pp. 186–200. Springer, Heidelberg (2004)
14. Legay, A., Delahaye, B., Bensalem, S.: Statistical model checking: An overview. In: Barringer, H., et al. (eds.) RV 2010. LNCS, vol. 6418, pp. 122–135. Springer, Heidelberg (2010)
15. Olfati-Saber, R.: Flocking for multi-agent dynamic systems: Algorithms and theory. IEEE Trans. on Automatic Control 51(3), 401–420 (2006)
16. Penczek, W., Wozna, B., Zbrzezny, A.: Bounded model checking for the universal fragment of ctl. Fundam. Inf. 51(1), 135–156 (2002)
17. Pnueli, A.: The temporal logic of programs. In: FOCS, pp. 46–57 (1977)
18. Pnueli, A., Zaks, A.: PSL model checking and run-time verification via testers. In: Misra, J., Nipkow, T., Sekerinski, E. (eds.) FM 2006. LNCS, vol. 4085, pp. 573–586. Springer, Heidelberg (2006)
19. Reynolds, C.W.: Flocks, herds and schools: A distributed behavioral model. In: Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH 1987), pp. 25–34. ACM (1987)
20. Stonedahl, F., Wilensky, U.: Finding forms of flocking: Evolutionary search in ABM parameter-spaces. In: Bosse, T., Geller, A., Jonker, C.M. (eds.) MABS 2010. LNCS, vol. 6532, pp. 61–75. Springer, Heidelberg (2011)
21. Vicsek, T., Czirók, A., Ben-Jacob, E., Cohen, I., Shochet, O.: Novel type of phase transition in a system of self-driven particles. Physical Review Letters 75, 1226–1229 (1995)
22. Younes, H.K.L.: Verification and Planning for Stochastic Processes. Ph.D. thesis, Carnegie Mellon (2005)

Blocking Advertisements on Android Devices Using Monitoring Techniques[⋆],[⋆⋆]

Khalil El-Harake, Yliès Falcone,
Wassim Jerad, Mattieu Langet, and Mariem Mamlouk

Laboratoire d'Informatique de Grenoble, Vérimag, University of Grenoble-Alpes, France
First.Last@imag.fr

Abstract. This paper explores the effectiveness and challenges of using monitoring techniques, based on Aspect-Oriented Programming, to block adware at the library level, on mobile devices based on Android. Our method is systematic and general: it can be applied to block advertisements from existing and future advertisement networks. We also present miAdBlocker, an industrial proof-of-concept application, based on this technique, for disabling advertisements on a per-application basis. Our experimental results show a high success rate on most applications. Finally, we present the lessons learned from this experience, and we identify some challenges when applying runtime monitoring techniques to real-world case studies.

1 Introduction

Smartphone usage has dramatically increased over the past decade, presently accounting for 57.6% of mobile devices. On mobile devices, Android [1], is the leading platform holding 78.4% of the market [2]. The downside is that the popularity of Android made it a primary target of adware: studies show that 49% of the applications on the market are bundled with at least one ad library [3]. It has also become common practice for application developers to bundle multiple advertisement libraries into their software.

The prevalence of adware, reduces device performance, detracts from user experience, significantly contributes to battery drain [4], and raises privacy concerns through the collection of sensitive information (such as user location) [5].

In this paper we present how monitoring techniques can be used to disable advertisements in Android applications. More particularly, we are interested in enforcement monitoring where a so-called enforcement monitor receives the sensitive events from the application under scrutiny and uses an internal decision procedure to determine whether each event should be allowed or not. Using Aspect-Oriented Programming [6] (AOP), we insert, at the bytecode-level, enforcement monitors that give users the ability to disable advertisements in Android on a per-application basis. Our technique minimally modifies a targeted application in the sense that only the initial behavior related to the display of advertisements is impacted while the rest of the host application functions normally.

[⋆] The work presented in this paper is partially funded by Institut Carnot LSI.

[⋆⋆] This paper is an academic study of the effectiveness of using monitoring techniques on a large-scale and challenging case study. By no means it should be seen as an attempt to actually suppress advertisements in applications nor to jeopardize the source of income of the actors involved in the Android ecosystem.

Unlike other methods that work by modifying the host operating system, our method works on an unrooted stock Android device. Our method also differs from similar solutions that perform bytecode transformation, and relies on custom security languages, or low-level transformation using intermediate representations of bytecode [7,8]. We rely on Aspect-Oriented Programming, via the standard AspectJ compiler [9], which developers are more likely to be familiar with. A detailed comparison with related work, can be found in Sec. 7.

Contributions. The contributions of this paper are to:

– Introduce the use of AOP as a method for disabling ads on Android applications;
– Present miAdBlocker, an end-user application, implementing the technique;
– Discuss results evaluating the approach in practice;
– Explore the limitations of using AOP to modify closed-source applications and block advertisements.

Paper Organization. The rest of this paper is structured as follows. Section 2 presents background notions. The method for suppressing advertisements is presented in Sec. 3. Section 4 presents miAdBlocker, our industrial proof-of-concept that implements the method presented in Sec. 3. miAdBlocker comprises i) a completely re-developed version of Weave Droid [10], ii) an aspect that allows to suppress advertisements in many Android applications, and adds user-oriented features. Section 5 presents our evaluation of the method on a sample of 860 popular Android applications retrieved "off-the-shelf" from Google Play. In Sec. 6, we discuss some of the issues (and possible counter measures) encountered when applying miAdBlocker to Android applications. Section 7 discusses related work. Finally, Sec. 8 presents some concluding remarks and open perspectives.

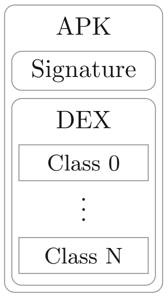
2 Background

This section presents some background notions on Android, advertisement libraries, aspect-oriented programming used in our approach.

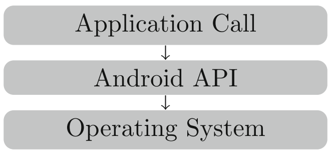
2.1 Android and Advertisement Librairies

Android is an open-source operating system based on Linux. Android is primarily used on mobile devices such as smartphones and tablets. Android applications are primarily developed in Java, and while it is possible to use native code for development, only 4.52% of applications on the market use it [11]. Unlike typical Java applications which run on a Java Virtual Machine (JVM), Android applications use the Dalvik Virtual Machine (DVM). The DVM and JVM have significant differences, such as differences in bytecode encoding, their differences and resulting problems are discussed briefly in Sec. 6.

Android applications are distributed as APK (Android Package) files (see Fig. 1(a)). APK files consist of the application's manifest, resources, application bytecode encoded for the DVM as a single `classes.dex` file, and signatures over the APK file for verifying its authenticity. An Android application runs in its own process, with its own



(a) Simplified structure of an APK.



(b) Android API call.

Fig. 1. General Information on the functioning of Android Application

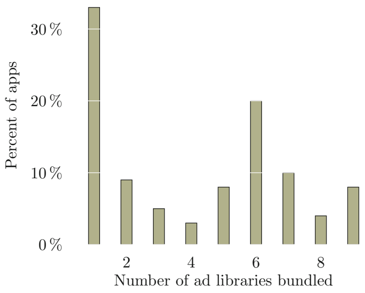


Fig. 2. Number of bundled advertisement libraries in applications

Dalvik Virtual Machine (DVM) instance. When a method call to a privileged resource is made, the call goes through the Android API, see Fig. 1(b), and the application framework checks if the originating application has the permission for proceeding with the request.

Advertisement libraries are often bundled with Android applications. An analysis of 100 applications containing advertisements on the market revealed that 40% of the applications contained 6 or more advertisement libraries, see Fig. 2. While another study, showed that 35% of applications contained 2 or more advertisement libraries [12].

Android does not have a built-in, in-process permission separation mechanism for libraries. As with all libraries bundled into an Android application, these advertisement libraries share the access permissions of the host application.

2.2 Aspect-Oriented Programming

Aspect-Oriented Programming (AOP) is an established paradigm developed in the 1990s at Xerox PARC [13]. AOP aims to facilitate modularity through the use of *aspects*, with each aspect being the embodiment of a so-called *cross-cutting concern*, i.e., parts of a program that rely on or must affect other parts of the system.

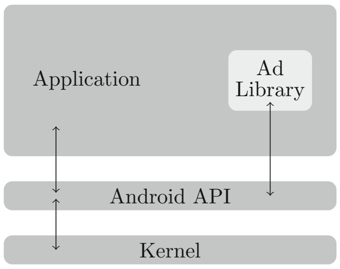


Fig. 3. View of an application at runtime with an ad library

Aspects are implemented through the use of three main concepts: joint points, pointcuts, and advices.

Joint-Point: A join point is an identifiable point during the runtime of a program, such as on the execution or call of a method.

PointCut: A pointcut is an expression for matching on joint points, for example the below pointcut matches on join points where a call to a method in the package `com.google.ads` is made.

```
call(* com.google.ads..*(..))
```

Advice: An advice is a piece of code that can be attached to run either after, before, or around a pointcut. For example, by using an *around* advice on a method, one can decide to proceed or not with the actual method call. Context information for making decisions regarding granting permissions can also be obtained by matching on the call arguments, or on other information via AspectJ's *thisJoinPoint* special variable. See Listing 1 for an example of an advice definition.

```
Object around() :
  call(* com.google.ads..*(..)) {
  return null;
}
```

Listing 1: Example of an around advice that can be used to block certain ads by intercepting calls to the `com.google.ads` package

AspectJ is an AOP implementation created at Xerox PARC for Java. The AspectJ compiler allows to perform weaving into JVM bytecode, through it, one can use aspects to modify compiled applications even without having access to the source code.

2.3 Weave Droid

Weave Droid [10] is a tool for weaving AspectJ aspects into an Android application. As input Weave Droid takes an APK, and a set of aspects that will be weaved into the APK. Weave Droid supports embedded weaving, where the entire weaving process is performed on the Android device, as well as cloud-based weaving, where the input

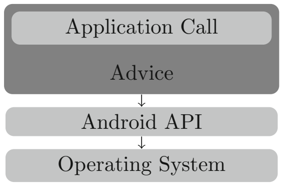


Fig. 4. Monitored Android API call

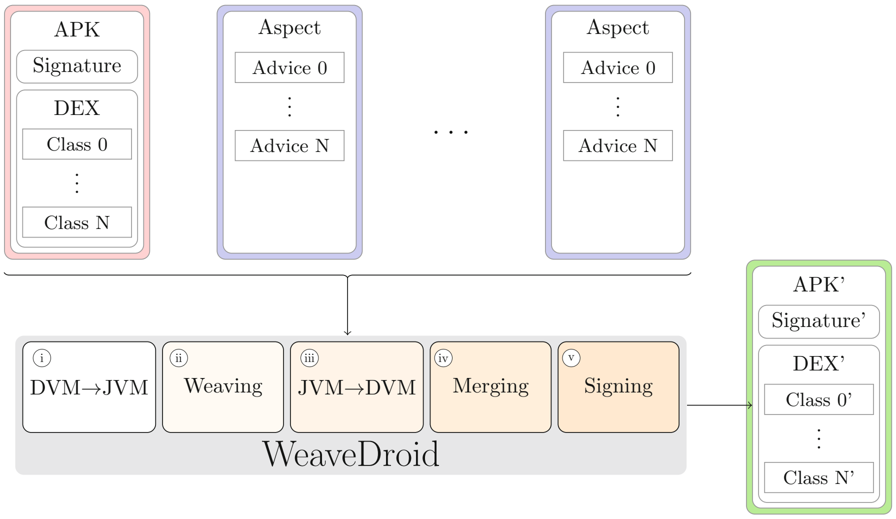


Fig. 5. Pipeline of weaving process

required for weaving is sent to be processed on a Weave Droid server after which the output is returned back to the device.

The Weave Droid process is split into 5 stages:

ⓘ Conversion of the input APK from DVM bytecode format to JVM format. This process uses the *dex2jar* library. This step is necessary as AspectJ is only capable of weaving JVM format bytecode. This conversion process has limitations that are discussed in Sec. 6.1.

ⓘⓘ Weaving of the input aspects with the JVM bytecode from stage ⓘ. The AspectJ compiler is used to handle the compilation and weaving of the aspects as well as injecting a library dependency required by the aspects.

ⓘⓘⓘ Conversion of the JVM bytecode to DVM bytecode format. This process uses the *dx* tool. This stage is necessary as Android expects the bytecode in DVM format.

ⓘⓥ Merging the modified bytecode into the input APK. From stage ⓘⓘⓘ we obtain a DVM bytecode file, we use this file to replace the `classes.dex` file present in the input APK.

ⓥ Signing of the modified APK. For Android applications to run, a valid signature is required. Modification of the APK from stage ⓘⓥ results in the APK's signatures being broken, to resolve this, we erase the existing signatures and re-sign the APK.

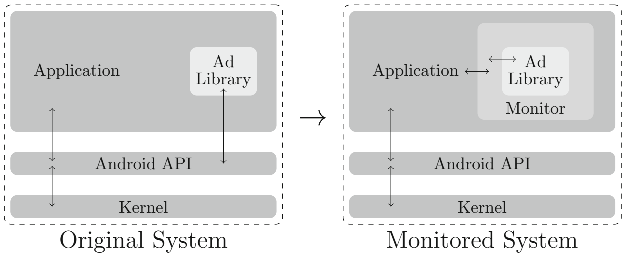


Fig. 6. Comparison of original application with the monitored application

The APK resulting from the Weave Droid process will exhibit the functionality introduced by the aspects, and will differ structurally from the original as follows:

- *Bytecode and size*, due to weaving of the aspects and inclusion of their library dependency.
- *Signature*, as a result of breaking the APK signature in stage (iv) and re-signing in stage (v).

3 Ad Suppression Method

In our solution we use the Weave Droid engine to weave the ad-blocking aspects into the application.

```
Class.forName("com.google.ads.AdActivity")
   .getDeclaredMethod("startActivity")
   .invoke(null);
```

Listing 2: Example of a method invocation via the reflection API

3.1 Aspect Creation

When writing aspects that modify the behavior of applications, we must take into account different mechanisms by which a method can be triggered. Adware applications are notorious for their use of dynamic method invocation as a means of defeating static analysis. For example through the reflection API a method call can be invoked. Listing 2 is an example call to a method that would not be intercepted by the pointcut example specified in Sec. 2.2.

Other factors that must be taken into account, include properly allocating and deallocating intercepted objects that require them. For example some objects such as *BroadcastReceiver* must be registered and unregistered.

Another issue which we must take care of is returning proper pseudo-objects in cases where we wish to spoof information such as contact lists, instead of simply blocking a method and returning null, which may cause the program to crash.

Listing 3, is a snippet of aspect code which blocks invocations to the method ”loadNewAd” within the ”com.inmobi.androidsdk” package, while allowing other method calls for the package to pass through. The snippet takes into account indirect calls via the reflection API, by wrapping an advice around calls to the ”java.lang.reflect.Method.invoke” method.

```
Object around() : execution(* com.inmobi.androidsdk.*.loadNewAd(..)) {
  return null;
}

Object around(): call(Object java.lang.reflect.Method.invoke(..)) {
  java.lang.reflect.Method target =
    (java.lang.reflect.Method)(thisJoinPoint.getTarget());
  Object[] args = thisJoinPoint.getArgs();
  if (args != null && args.length > 0 && args[0] != null) {
    String receiver = args[0].getClass().getName();
    if (target.getName().compareTo("loadNewAd") == 0
    && receiver.startsWith("com.inmobi.androidsdk"))
    return;
  }
  return proceed();
}
```

Listing 3: Shortened example of aspect code for blocking inmobi ads

3.2 Amending the Application

In this section we describe the steps taken by our implementation for suppressing the advertisements of an input application (see Fig. 7).

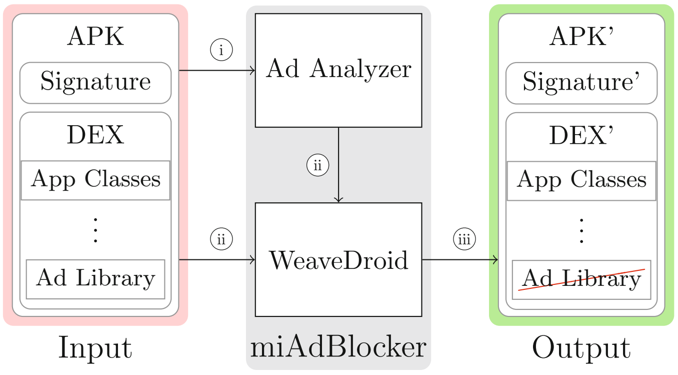
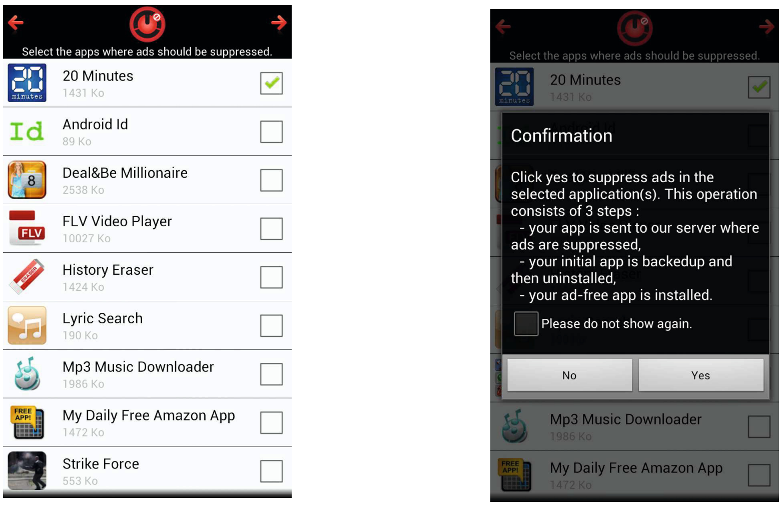


Fig. 7. Pipeline of advertisement suppression process

ⓘ The application’s APK file is passed to the Ad Analyzer. The Ad Analyzer searches through the libraries used by the application and compares them to a list of known advertisement network libraries. Using this list the Ad Analyzer determines the set of aspects to use for ad blocking.


Select the apps where ads should be suppressed.

20 Minutes 1431 Ko	✓
Android Id 89 Ko	☐
Deal&Be Millionaire 2538 Ko	☐
FLV Video Player 10027 Ko	☐
History Eraser 1424 Ko	☐
Lyric Search 190 Ko	☐
Mp3 Music Downloader 1986 Ko	☐
My Daily Free Amazon App 1472 Ko	☐
Strike Force 553 Ko	☐

(a) Showing the list of applications harboring advertisement libraries.

Confirmation

Click yes to suppress ads in the selected application(s). This operation consists of 3 steps :
- your app is sent to our server where ads are suppressed,
- your initial app is backedup and then uninstalled,
- your ad-free app is installed.

☐ Please do not show again.

No | Yes

(b) Confirmation before processing an application.

Fig. 8. miAdBlocker in action

ⓘⓘ The application's APK file, and the set of aspects required for blocking the advertisements specific to it are passed to Weave Droid, which may be remote or local. Weave Droid handles weaving of the aspects into the application.
ⓘⓘⓘ Finally, Weave Droid outputs a new APK. The output APK contains the ad blocking behavior in it, and therefore fail to display ads.

4 Implementation: miAdBlocker

miAdBlocker, is a user-friendly Android application based on the methodology described in Sec. 3. miAdBlocker is implemented using Java in 7,260 lines of code (LoC), and uses a remote Weave Droid server for weaving enforcement monitors inside applications. The application allows users to selectively disable the advertisements of applications installed on their device. The implementation supports devices running on Android version 2.3.3 and higher. It uses a 2,190 LoC aspect library capable of disabling over 30 different ad network libraries.

At startup miAdBlocker scans all the applications installed on the system; detecting for the presence of advertisement libraries. Then, a list is populated with all the applications that are candidates for ad-blocking, as seen in Fig. 8(a). The user may select the application for which he wants to block ads. Once the user has indicated that he wants ads to be removed from the application, a confirmation dialog window is displayed as seen in Fig. 8(b).

While weaving directly on the device is possible, due to the inherent limitations and performance issues of weaving aspects directly on android devices [10], miAdBlocker defaults to querying a Weave Droid server, to handle the process.

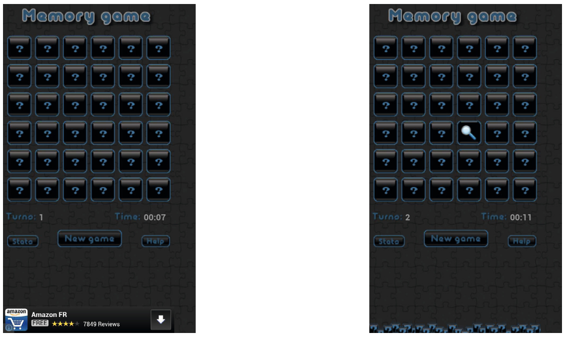


Fig. 9. An application before (left) and after (right) being processed by miAdBlocker

5 Evaluation

This section presents our evaluation of miAdBlocker with "off-the-shelf" Android applications retrieved from Google Play.

5.1 Case Study

We ran and experiment focused on analyzing the reliability of miAdBlocker to amend applications with ad-blocking enforcement monitors, while preserving the features of the target application (the application should remain functionable and its performance should not be degraded).

To evaluate the proposed method of ad-blocking in Android applications, we tested a sample of 860 popular applications from different categories (games, utilities, misc), and recorded the success or failure of an application.

There are three phases to our testing process. If any error occured during a phase, the application was considered to have failed the current tests, and the tests after it.

Applications that were successfully modified, had their modified versions put through the execution test. Due to the time consuming nature of thoroughly testing applications, for the third test a randomized sample of applications from those that were successful in the execution test were selected for this stage.

Modification. Amending the application with ad-blocking aspects and repackaging it.
Execution. Installing, initial launch, and uninstallation of the amended application.
Thorough. All activity windows of an application were checked to ensure proper functioning.

Table 1 shows the results of the three test phases. For each phase we show the number of applications tested, and their success rate. In each of the phases we encountered errors which we explain below.

Modification involves invoking the Weave Droid pipeline which consists of several sub-steps as seen in Fig. 5. In the first step we have to perform DVM$\rightarrow$JVM bytecode retargeting, this process is error prone and discussed in Sec. 6.1. We also encountered

Table 1. Number of applications tested and their success rates at each of the stages

	Modification		Execution		Thorough	
Games	341	96.19%	328	85.98%	52	77.61%
Utility	95	98.95%	94	96.81%	52	94.12%
Misc	424	97.41%	413	93.22%	30	100%

applications exploiting limitations in the retargeting tool dex2jar, for example by using method and field names obfuscated with unicode characters dex2jar crashes. During the weaving phase AspectJ encountered "missing type" errors due to the presence of calls, and type references that do not have their corresponding libraries bundled with the application.

In the execution and thorough testing phase, we had crashes due to the introduction of errors in the modification phase. We also encountered errors possibly due to the presence of anti-tampering code, and found that game applications in particular had a much higher rate of failure at execution than other application categories.

Remark 1 (A note on performance and file size overhead). The aspects applied to the programs affect their performance. However as with the work done previously we found that the types of aspects developed for ad-blocking had a negligible effect on performance degredation [10]. Rather we observed performance improvements, bandwidth savings, and energy savings. Indeed, our aspects are woven at compile time, and disable a significant amount of code from being run.

The amendment process requires the inclusion of a fixed 117KB library, as well as the newly woven classes/aspects bytecode. A review of the APK sizes before and after the ad-blocking transformation, showed that there was a negligible increase in APK size in the range of 0-5%, and overall averaged near 0%.

6 Discussion

While there are currently many challenges faced from a bytecode weaving approach, there remain advantages, in that it is a more targeted approach and unlike other approaches it does not require alteration of the host operating system, nor use of superuser permissions, both of which may result in voiding of the device warranty. This section discusses some of the challenges faced (and possible counter measures) when applying runtime monitoring and miAdBlocker to Android applications.

Analysis of 100 applications on the market revealed that the majority of those tested used features such as encryption and classloaders. These features may be used to circumvent methods relying on static analysis and bytecode weaving. Further implications of using these features are discussed in the rest of this section. We believe these issues were not raised by previous monitoring frameworks applied to academic benchmarks. These issues are to be considered when using a monitoring framework for third-party applications. See Table 2 for the results of the analysis.

Table 2. Some features used by 100 free applications with advertisements from the top 200 on Google Play

Feature	Percent of Apps Tested
Reflection	99%
Encryption	96%
ClassLoader	99%
Native Code	0%
Calls External Executable	88%

6.1 DVM to JVM Retargeting

There are significant differences between the register-based Dalvik Virtual Machine and the stack-based Java Virtual Machine, these differences result in information loss when converting bytecode from one format to the other. The information loss is a cause for some of the errors we encounter when running our tool-chain. As resolving this issue is an active area of research [14], in time we expect success rates to increase.

Malware has also been known to take advantage of bugs in present in retargeting software, preventing proper conversion [15]. Modifying the AspectJ compiler to target Dalvik bytecode directly is a possible solution for avoiding the problems introduced by intermediate retargeting software.

6.2 Native Code

Our method is limited to the modification of Java-based applications, and may be bypassed in applications using native code. But as stated earlier, due to the difficulties of developing apps using native code, only a small percentage of the available applications available use it, and even the ones that use it only use it in small critical performance areas.

6.3 Tamper Detection

Applications using tamper detection can detect unauthorized modification. Developers can integrate this detection using tools such as Arxan [16] and Google LVL [17]. Upon the detection of tampering, applications may be designed to exit, or behave improperly.

Detection typically revolves around signature verification. Application modification as a side-effect results in different signatures compared to the original application.

As miAdBlocker and Weave Droid by design modify the application package, they fall prey to this detection; contributing to the failure rates seen during application execution. Bypassing this mechanism would allow for higher success rates. We will briefly discuss countermeasures for two basic common techniques, of implementing this detection.

Package signature verification. Applications are signed by the developers using a private key that is only accessible by them. When an application is modified, the original signature will no longer correspond to it. An application without a valid signature will fail to run. Thus, to have a usable application the tamperer must sign it with their own key. A detection mechanism could be for the application to compare its current signature against a copy of the signature known to be authentic.

```
Signature[] sig =
  getPackageManager()
  .getPackageInfo(app, PackageManager.GET_SIGNATURES)
  .signatures;

if (sig[0].hashCode() != authenticSignature) fail();
```

Listing 4: Example implementation of tamper detection

A countermeasure could be to store the valid package signature before transformation, and to intercept package manager calls in the modified application. The modified application would return the original recorded application's signature, thus passing this test.

File signature verification. File signature verification is another form of protection used by application developers. The method involves computing a checksum value of the application files, using a hash function. Detection can be performed at application startup by recomputing the checksum values of the current files and comparing them against the previously computed authentic hashes.

Counter-measures could be to:

– Keep a copy of the unmodified application, and intercepting Java's file system libraries. When an application wishes to access its own files, the interception method redirects access to the original versions that would pass the signature checks.
– Intercept common hash functions used for signature checking, and return a precomputed correct hash upon request.

Amending applications with more complicated verification systems can be done by integrating verification library subversion tools into the Weave Droid pipeline.

6.4 Dynamically Loaded Code

Java allows for code not present in the application to be loaded at runtime from either a local path, or an online location, using a *ClassLoader*. As this code is not present for Weave Droid to perform transformations on, the dynamically loaded code is free of the behaviors enforced upon the rest of the application. Developers may use this mechanism to dynamically load advertisements, bypassing ad blocking utilities based on static analysis and bytecode transformation.

Our technique can be extended to handle such cases via interception of calls to the ClassLoader. A custom ClassLoader can then analyze and send the code to a Weave Droid server. There, the desired behavior is enforced on the code, then returned to the device for execution.

Table 3. Comparison of the requirements of several ad-blocking applications

	Requirements		
	Root	Proxy	Reboot
Adblock plus	✓	✓	✗
AirPush Block	✓	✗	✗
Adway	✓	✗	✗
MyInternetSecur	✗	✗	✓
MiAdBlocker	✗	✗	✗

6.5 Obfuscation

Obfuscation is a technique employed by developers to protect their applications against reverse engineering and analysis. Through obfuscation, the names of methods, classes and packages are rewritten while preserving the functionality of the app. Tools such as ProGuard fulfill this purpose. This technique renders aspects that would have worked, targeting specific pointcuts based on names unusable.

However, we did not encounter much problems in this regard when blocking ad libraries, as it is common practice to preserve the public APIs of said libraries due to issues arising from their obfuscation.

Another type of obfuscation can be performed involves storing the code in encrypted form, this code is then decrypted and loaded by a ClassLoader at runtime. Our solution of intercepting the ClassLoader would properly account for this problem, as the ClassLoader must take in the unencrypted bytecode.

6.6 Signature Modification

A side-effect of this modification is that market updates are not properly detected for the modified application, and if one wishes to update directly from the market, they would have to first uninstall or restore the application, before they can move to a newer version. To solve this issue a separate mechanism must be used to check and handle updates.

7 Related Work

7.1 Comparison with Ad-Blocking Software on the Market

We made a survey of the ad blocking solutions found on Google Play and compiled the results (see Table 3). Compared to the other tools on the market that were surveyed, miAdBlocker had less requirements for enforcement of ad blocking, making it more user-friendly.

7.2 Comparison with Similar Research Projects

miAdBlocker [10], was extended upon. The Weave Droid engine is now reimplemented in 6,130 lines of Java code, with a focus on robustness. Originally Weave Droid only handled Google ads; with miAdBlocker we can handle over 30 different advertisement networks.

Aurasium [18] is a policy enforcer that intercepts Android applications via a native library layer. Unlike their method, our method has the benefit of using monitors with awareness of the call context in Java, giving us the advantage of selectively enforcing monitors on a finer-grained level (such as per library level).

Bartel, et al. [8] present a method and implementation that performs static analysis. Our system differs in that we can make decisions dynamically with the awareness of context and avoids the false positives usually induced when using static analysis. For instance, our tool can detect specific (dynamically computed) URLs instead of only detecting that an HTTP connection is made in the application.

8 Conclusion and Future Work

8.1 Conclusion

This paper studies the use of monitoring techniques on a real application scenario: blocking advertisement on third-party Android applications retrieved "off-the-shelf" from public repositories such as Google Play. Our purpose was to produce a tool that goes beyond usual research prototypes in runtime verification as we aimed to reach a level of maturity allowing our tool to be publicly released and delivered to users without any computer-science background. During our case studies we encountered many challenges such as the number and heterogeneity of Android applications on which our technique has to be tested, the diversity of the possibilities for developers to displaying advertisements, the discrepancy between the Dalvik format (executable) and the bytecode format (instrumentable). The challenges stem from the facts that we target third-party applications, from many developers, in a domain where a strong competition exists between applications.

We use good practices obtained from research endeavors in the runtime verification community and encode monitoring using Aspect-Oriented Programming. Our experiments show that using AOP via AspectJ is an effective technique to modify existing closed source applications to incorporate ad-blocking monitors. Unlike tools relying on low-level bytecode analysis and transformation, AOP allows for easier targeting and modification of existing application code; through specification of transformation sites via a pointcut matching system. Our method also has the benefit over other solutions in that it has been implemented and tested to work embedded from an android device, and as a cloud-based service.

Our experiments showed a good success rate overall, with better success rates depending on the category of the application. Analysis of applications on the market however, showed the heavy presence of features that could be used for circumvention of the enforcement mechanism. These features are targeted in a newer implementation.

8.2 Future Work

Even if our approach is dedicated to blocking advertisement on Android applications the challenges encountered in this paper will remain when applying monitoring in other application domains sharing features (e.g., third-party applications). Thus, we believe that future research endeavors in the runtime verification community should consider extending monitoring techniques to deal with the issues of dynamically loaded code, obfuscation, and tamper-resistant code, for the purpose of yielding a higher success rate at integrating effective monitors.

References

1. Google Inc.: Android (2014), `http://www.android.com`, `http://developer.android.com`
2. Gartner: Market share analysis: Mobile phones, worldwide, 4q13 and 2013 (2013)
3. Pearce, P., Felt, A.P., Nunez, G., Wagner, D.: Addroid: Privilege separation for applications and advertisers in android. In: Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security, pp. 71–72. ACM (2012)
4. Pathak, A., Hu, Y.C., Zhang, M.: Where is the energy spent inside my app?: Fine grained energy accounting on smartphones with eprof. In: Proceedings of the 7th ACM European Conference on Computer Systems, pp. 29–42 (2012)
5. Stevens, R., Gibler, C., Crussell, J., Erickson, J., Chen, H.: Investigating user privacy in android ad libraries. In: Proceedings of Mobile Security Technologies Workshop, MoST (2012)
6. Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J.M., Irwin, J.: Aspect-oriented programming. In: Akşit, M., Matsuoka, S. (eds.) ECOOP 1997. LNCS, vol. 1241, pp. 220–242. Springer, Heidelberg (1997)
7. Backes, M., Gerling, S., Hammer, C., Maffei, M., von Styp-Rekowsky, P.: AppGuard – enforcing user requirements on android apps. In: Piterman, N., Smolka, S.A. (eds.) TACAS 2013. LNCS, vol. 7795, pp. 543–548. Springer, Heidelberg (2013)
8. Bartel, A., Klein, J., Monperrus, M., Allix, K., Traon, Y.L.: Improving privacy on android smartphones through in-vivo bytecode instrumentation. CoRR abs/1208.4536 (2012)
9. Xerox Corporation: Aspectj programming guide (2014), `http://www.eclipse.org/aspectj/`
10. Falcone, Y., Currea, S.: Weave Droid: aspect-oriented programming on Android devices: fully embedded or in the cloud. In: Goedicke, M., Menzies, T., Saeki, M. (eds.) ASE, pp. 350–353. ACM (2012)
11. Zhou, Y., Wang, Z., Zhou, W., Jiang, X.: Hey, you, get off of my market: Detecting malicious apps in official and alternative android markets. In: NDSS. The Internet Society (2012)
12. Shekhar, S., Dietz, M., Wallach, D.S.: Adsplit: Separating smartphone advertising from applications. In: USENIX (2012)
13. Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J.M., Irwin, J.: Aspect-oriented programming. In: Akşit, M., Matsuoka, S. (eds.) ECOOP 1997. LNCS, vol. 1241, pp. 220–242. Springer, Heidelberg (1997)
14. Octeau, D., Jha, S., McDaniel, P.: Retargeting android applications to java bytecode. In: Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering, p. 6. ACM (2012)
15. Chenette, S.: Building custom android malware, BruCON (2013)
16. Arxan: Ensureit® for android on arm (2013)
17. Google Inc.: Licensing overview - android developers (2014)
18. Xu, R., Saïdi, H., Anderson, R.: Aurasium: Practical policy enforcement for android applications. In: Proceedings of the 21st USENIX Conference on Security Symposium, Security 2012, p. 27. USENIX Association, Berkeley (2012)

Monitoring with Data Automata

Klaus Havelund[⋆]

Jet Propulsion Laboratory
California Institute of Technology
California, USA

Abstract. We present a form of automaton, referred to as *data automata*, suited for monitoring sequences of data-carrying events, for example emitted by an executing software system. This form of automata allows states to be parameterized with data, forming named records, which are stored in an efficiently indexed data structure, a form of database. This very explicit approach differs from other automaton-based monitoring approaches. Data automata are also characterized by allowing transition conditions to refer to other parameterized states, and by allowing transitions sequences. The presented automaton concept is inspired by rule-based systems, especially the RETE algorithm, which is one of the well-established algorithms for executing rule-based systems. We present an optimized external DSL for data automata, as well as a comparable unoptimized internal DSL (API) in the SCALA programming language, in order to compare the two solutions. An evaluation compares these two solutions to several other monitoring systems.

1 Introduction

Runtime verification (RV) is a sub-field of software reliability focused on how to monitor the execution of software, checking that the behavior is as expected, and if not, either produce error reports or modify the behavior of the software as it executes. The executing software is instrumented to emit a sequence of events in some formalized event language, which is then checked against a temporal specification by the monitor. This can happen during test before deployment, or during deployment in the field. Orthogonally, monitoring can occur online, simultaneously with the running program, or offline by analyzing log files produced by the running program. Many RV systems have appeared over the last decade. The main challenges in building these systems consist of defining expressive specification languages, which also makes specification writing attractive (simple properties should have simple formulations), as well as implementing efficient monitors for such. A main problem is how to handle data-carrying events efficiently in a temporal setting. Consider for example the following event stream consisting of three $grant(t, r)$ events (resource r is granted to task t):

[⋆] The work described in this publication was carried out at Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

$\langle grant(t_1, a), grant(t_2, b), grant(t_3, a)\rangle$, and consider the property that no resource should be granted to more than one task at a time. When receiving the third event $grant(t_3, a)$, the monitor has to search the relevant history of seen events, which, if one wants to avoid looking at the entire history, in the presence of data ends up being a data indexing problem in some form or another.

RV systems are typically based on variations of state machines, regular expressions, temporal logics, grammars or rule-based systems. Some of the most efficient RV systems tend to be limited wrt. expressiveness [2], while very expressive systems tend to not be competitive wrt. efficiency. Our earlier work includes studies of rule-based systems, including RULER [6] and LOGFIRE [15]. As example of a rule in a rule-based system, consider: $Granted(t, r) \wedge grant(t', r) \Rightarrow Error(t, t', r)$. The state of a rule-system can abstractly be considered as consisting of a set of *facts*, referred to as the *fact memory*, where a fact is a named data record, a mapping from field names to values. A fact represents a piece of observed information about the monitored system. A condition in a rule's left-hand side can check for the presence or absence of a particular fact, and the action on the right-hand side of the rule can add or delete facts. Left-hand side matching against the fact memory usually requires unification of variables occurring in conditions. In case all conditions on a rule's left-hand side match (become true), the right-hand side action is executed. The rule above states that if the fact memory contains a fact that matches $Granted(t, r)$ for some task t and resource r, and a $grant(t', r)$ event is observed, then a new fact $Error(t, t', r)$ is added to the fact memory. A well-established algorithm for efficiently executing rule-based systems is the RETE algorithm [12], which we implemented in the LOGFIRE system [15] as an internal DSL (API essentially) in the SCALA programming language, while adopting it for runtime verification (supporting events in addition to facts), and by optimizing fact search using indexing.

While an interesting solution, the RETE algorithm is complex. Our goal is to investigate a down-scaled version of RETE to an automaton-based formalism, named *data automata* (DAUT), specifically using the indexing approach implemented in [15]. Two alternative solutions are presented and compared. First, data automata are presented as a so-called *external DSL*, a stand-alone formalism, with a parser and interpreter implemented in SCALA. The formalism has some resemblance to process algebraic notations, such as CSP and CCS. Second, we present an unoptimized *internal DSL* (DAUT^{int}), an API in the SCALA programming language, with a very small implementation, an order of magnitude smaller compared to the external DSL (included in its entirety in Appendix A). An internal DSL has the advantage of offering all the features of the host programming language in addition to the features specific to the DSL itself. We compare these two solutions with a collection of other monitoring systems.

The paper is organized as follows. Section 2 outlines related work. Section 3 presents data automata, as the external DSL named DAUT, including their pragmatics, syntax, and semantics. Section 4 presents an indexing approach to obtain more efficient monitors for data automata. Section 5 presents the alternative internal SCALA DSL named DAUT^{int}, which also implements the data automaton concept. Section 6 presents an evaluation, comparing performance with other systems. Section 7 concludes the paper.

2 Related Work

The inspiration for this work has been our work on the rule-based LogFire system [15], which again was inspired by the Ruler system [6]. The external DSL is closely related to LogScope [4]. The internal Scala DSL is a modification of the internal Scala DSL TraceContract [5]. As such this work can be seen as presenting a reflection of these four pieces of work.

The first systems to handle parameterized events appeared around 2004, and include such systems as Eagle [3] (a form of linear μ-calculus), Jlo [18] (linear temporal logic), TraceMatches [1] (regular expressions), and Mop [16] (allowing for multiple notations). Mop seems the most efficient of all systems. The approach applied is referred to as *parametric trace slicing*. A trace of data carrying events is, from a semantic point of view, sliced to a set of propositional traces containing propositional events, not carrying data (one trace for each binding of data parameters) which are then fed to propositional monitors. In practice, however, the state of a monitor contains, simplified viewed, a mapping from bindings of parameter values to propositional monitor states. This indexing approach results in an impressive performance. However, this is at the price of some lack of expressiveness in that properties cannot relate different slices, as also pointed out in [2]. MopBox [10] is a modular Java library for monitoring, implementing Mop's algorithms.

Quantified Event Automata [2] is an automaton concept for monitoring parameterized events, which extends the parametric trace slicing approach used in Mop by allowing event names to be associated with multiple different variable lists (not allowed in Mop), by allowing non-quantified variables to vary during monitoring, and by allowing existential quantification in addition to universal quantification. This results in a strictly more expressive logic. This work arose from an attempt to understand, reformulate and generalize parametric trace slicing, and more generally from an attempt to explore the spectrum between Mop and more expressive systems such as Eagle and Ruler, similar to what is attempted in the here presented work. The work is also closely related to Orchids [13], which is a comprehensive state machine based monitoring framework created for intrusion detection.

Several systems have appeared that monitor first order extensions of propositional linear temporal logic (LTL). A majority of these are inspired by the classical rules (Gerth et. al) for rewriting LTL. These extensions include [17], an extension of LTL with a binding operator, and implemented using alternating automata; LTL-FO^{+} [14], for parameterized monitoring of Xml messages communicated between web-services; Mfotl [7], a metric first-order temporal logic for monitoring, with time constraints as well as universal and existential quantification over data; LTL$^{\mathrm{FO}}$ [8], based on spawning automata; and [11], which uses a combination of classical monitoring of propositional temporal properties and SMT solving.

Listing 1. Monitor for requirements R_1 and R_2

```
monitor R1R2 {
  init always Start {
    grant(t, r) → Granted(t,r)
    release(t, r) :: ¬Granted(t,r) → error
  }

  hot Granted(t,r) {
    release(t,r) → ok
    grant(_,r) → error
  }
}
```

3 The Daut Calculus

3.1 Illustration by Example

We shall introduce Daut by example. Consider a scenario where we have to write a monitor that monitors sequences of $grant(t, r)$ and $release(t, r)$ events, representing respectively granting a resource r to a task t, and task t releasing resource r. Consider furthermore the two requirements R_1: *"a grant of a resource to a task must be followed by a release of that resource by the same task, without another grant of that resource in between (to the same task or any other task)"*, and R_2: *"a resource cannot be released by a task, which has not been granted the resource"*. These requirements can be formalized in Daut as shown in Listing 1. The monitor has the name *R1R2*. It contains two states *Start* and *Granted*, the latter of which is parameterized with a task t and a resource r. The *Start* state is the initial state indicated by the modifier **init**. Furthermore, it is an **always** state, meaning that whenever a transition is taken out of the state, an implicit self-loop keeps the state around to monitor further events. In the *Start* state when a $grant(t, r)$ event is observed, a $Granted(t, r)$ state is created. If a $release(t, r)$ event is observed, and the condition occurring after :: is *true*, namely that there is no $Granted(t, r)$ state active, it is an error. The state $Granted(t, r)$ is a so-called hot state, which essentially is a non-final state. It is an error to remain in a hot state at the end of a log analysis.

The formalism allows for various abbreviations. For example, it is possible to write transitions at the top level, as a shorthand for introducing a state with modifiers **init** and **always**. This is illustrated by the monitor in Listing 2, which is semantically equivalent to the monitor in Listing 1. Also, target states can be "inlined", making it possible to write sequences of transitions without mentioning intermediate states. This is a shorthand for the longer form where each intermediate state is named. As an example, requirement R_1 can be stated

Listing 2. Simplified monitor

```
monitor R1R2 {
  grant(t, r) → Granted(t,r)
  release(t, r) :: ¬Granted(t,r) → error

  hot Granted(t,r) {
    release(t,r) → ok
    grant(_,r) → error
  }
}
```

Listing 3. Monitor for requirement R_1

```
monitor R1 {
  grant(t, r) → hot {
    release(t,r) → ok
    grant(_,r) → error
  }
}
```

succinctly as shown in Listing 3. Such nesting can be arbitrarily deep, corresponding to time lines. This makes it possible to write monitors that resemble temporal logic, as also was possible in TraceContract [5].

In general, states can be parameterized with arbitrary values represented by expressions in an expression language (not just identifiers as in some RV approaches, for example Mop). The formalism allows counting, as an example. The right-hand sides of transitions can for brevity also be conditional expressions, where conditions can refer to state and event parameters, as well as other states. To summarize, this automaton concept supports parameterized events, parameterized states, transition conditions involving state and event parameters as well as other parameterized states, expressions as arguments to states, and conjunction of conditional target states. What is not implemented from classical rule-based systems is disjunction of target states (as in Ruler), variables and general statements as actions, deletion of facts in general (only the state from which a transition leads is deleted when taking the transition, except if it is an **always** state), and general unification across conditions. A further extension of this notation (not pursued in this work) could allow declaration of variables local to a monitor, reference to such in conditions, as well as arbitrary statements with side-effects on these variables in right-hand side actions. The internal Scala DSL Daut^{int} presented in Section 5 does support these extensions.

3.2 Syntax

The presentation of data automata shall focus on the syntax of such, as used in the specifications seen in the previous subsection. The full grammar for DAUT is shown in Figure 1, using extended BNF notation, where $\langle N\rangle$ denotes a non-terminal, $\langle N\rangle$ ::= ... defines the non-terminal $\langle N\rangle$, S^* denotes zero or more occurrences of S, S^{**} denotes zero or more occurrences of S separated by commas (','), $S \mid T$ denotes the choice between S and T, $\lceil S\ \rceil$denotes optional S, **bold** text represents a keyword, and finally '...' denotes a terminal symbol.

$\langle \mathit{Specification}\rangle$ ::= $\langle \mathit{Monitor}\rangle^*$

$\langle \mathit{Monitor}\rangle$ ::= **monitor** $\langle \mathit{Id}\rangle$ '{' $\langle \mathit{Transition}\rangle^*$ $\langle \mathit{State}\rangle^*$ '}'

$\langle \mathit{State}\rangle$ ::= $\langle \mathit{Modifier}\rangle^*$ $\langle \mathit{Id}\rangle$ $\lceil$ ($\langle \mathit{Id}\rangle^{**}$) $\rceil$ $\lceil$ '{' $\langle \mathit{Transition}\rangle^*$ '}' $\rceil$

$\langle \mathit{Modifier}\rangle$ ::= **init** | **hot** | **always**

$\langle \mathit{Transition}\rangle$ ::= $\langle \mathit{Pattern}\rangle$ ':\:' $\langle \mathit{Condition}\rangle$ '$\rightarrow$' $\langle \mathit{Action}\rangle^{**}$

$\langle \mathit{Pattern}\rangle$ ::= $\langle \mathit{Id}\rangle$ '('$\langle \mathit{Id}\rangle^{**}$')'

$\langle \mathit{Condition}\rangle$ ::= $\langle \mathit{Condition}\rangle$ '$\wedge$' $\langle \mathit{Condition}\rangle$
  | $\langle \mathit{Condition}\rangle$ '$\vee$' $\langle \mathit{Condition}\rangle$
  | '$\neg$' $\langle \mathit{Condition}\rangle$
  | '('$\langle \mathit{Condition}\rangle$')'
  | $\langle \mathit{Expression}\rangle$ $\langle \mathit{relop}\rangle$ $\langle \mathit{Expression}\rangle$
  | $\langle \mathit{Id}\rangle$ $\lceil$ '('$\langle \mathit{Expression}\rangle^{**}$')' $\rceil$

$\langle \mathit{Action}\rangle$ ::= **ok**
  | **error**
  | $\langle \mathit{Id}\rangle$ $\lceil$ '('$\langle \mathit{Expression}\rangle^{**}$')' $\rceil$
  | **if** '(' $\langle \mathit{Condition}\rangle$ ')' **then** $\langle \mathit{Action}\rangle$ **else** $\langle \mathit{Action}\rangle$
  | $\langle \mathit{Modifier}\rangle^*$ '{' $\langle \mathit{Transition}\rangle^*$ '}'

Fig. 1. Syntax of DAUT

The syntax can briefly be explained as follows. A $\langle \mathit{Specification}\rangle$ consists of a sequence of monitors, each representing a data automaton. A $\langle \mathit{Monitor}\rangle$ has a name represented by an identifier $\langle \mathit{Id}\rangle$, and a body enclosed by curly brackets. The body contains a sequence of transitions and a sequence of states. The transitions are short for an **init**ial **always** state containing these transitions. A $\langle \mathit{State}\rangle$ is prefixed with zero or more modifiers (**init**, **always**, or **hot**), has a name, and an optional list of (untyped) formal parameters, and an optional body of transitions leading out of the state. A $\langle \mathit{Transition}\rangle$ consists of a pattern that can match (or not) an incoming event, where already bound formal parameters must match the parameters of the event, followed by a condition. If the pattern matches and the condition evaluates to *true*, the action is executed, leaving the enclosing state unless it is an **always** state. A $\langle \mathit{Condition} \rangle$ conforms to the standard Boolean format including relations over values of expressions. The last alternative $\langle \mathit{Id} \rangle$ ⌈ '('$\langle \mathit{Expression} \rangle$**')'⌉ allows to write state expressions as conditions. A state expression of the form $\mathit{id}(\mathit{exp}_1, \ldots, \mathit{exp}_n)$ is true if there is a state active with parameters equal to the value of the expressions. This specifically allows to express past time properties. An $\langle \mathit{Action} \rangle$ is either **ok**, meaning the transition is taken without further action (a skip), **error**, which causes an error to be reported, the creation of a new state (target state), a conditional action, useful in practice, or the derived form of a modifier-prefixed block of transitions, avoiding to name the target state.

3.3 Semantics

Basic Concepts. The semantics is defined as an operational semantics. We first define some basic concepts. We shall assume a set Id of identifiers and a set V of values. An environment $\mathit{env} \in \mathit{Env} = \mathit{Id} \overset{m}{\to} V$ is a finite mapping from identifiers to values. An event $e \in \mathit{Event} = \mathit{Id} \times V^*$ is a tuple consisting of an event name and a list of values. We shall write an event $(\mathit{id}, \langle v_1, \ldots, v_n \rangle)$ as: $\mathit{id}(v_1, \ldots, v_n)$. A trace $\sigma \in \mathit{Trace} = \mathit{Event}^*$ is a list of events. A state identifier id is associated with a sequence of formal parameters $\mathit{id}_1, \ldots, \mathit{id}_n$. A particular state $s \in \mathit{State} = \mathit{id}(v_1, \ldots, v_n)$, for $v_1, \ldots, v_n \in V$, represents an instantiation of the formal parameters. For such a state we can extract the environment with the following notation: $s.\mathit{env}$ of type Env, formed from the binding of the formal parameter ids to the values: $s.\mathit{env} = [\mathit{id}_1 \mapsto v_1, \ldots, \mathit{id}_n \mapsto v_n]$.

The semantics of each single monitor in a specification is a labeled transition system: $\mathit{LTS} = (\mathit{Config}, \mathit{Event}, \rightarrow, i, F)$. Here $\mathit{Config} \subseteq \mathit{State}$ is the set of all possible states (possibly infinite depending on the value domain). Event is a set of parameterized events. $\rightarrow \subseteq \mathit{Config} \times (\mathit{Event} \times \mathbb{B}) \times \mathit{Config}$ is a transition relation, which defines transitions from a configuration to another as a result of an observed event, while "emitting" a Boolean flag being *false* iff. an error has been detected. $i \subseteq \mathit{Config}$ is the set of initial states, namely those with modifier **init** (these cannot have arguments). Finally, $F \subseteq \mathit{Config}$ is the set of final states $\mathit{id}(v_1, \ldots, v_n)$ where id is not declared with modifier **hot**.

The operational semantics to be presented defines how a given configuration con evolves to another configuration con' on the observation of an event e. In addition, since such a move can cause an **error** state to be entered, a Boolean flag, the *status flag*, will indicate whether such an error state has been entered in that particular transition. The result of transitions will hence be pairs of the form $(\mathit{flag}, \mathit{con}) \in \mathit{Boolean} \times \mathit{Config}$, also called *results* (res). Furthermore, we shall use the value $\perp$ to indicate that an evaluation has failed, for example if no transitions are taken out of a state. Consequently we need to be able to compose results, potentially being $\perp$, where combination of two proper results is again a result consisting of the conjunction of flags and union of configurations. We define two operators, $\oplus_\perp$ (for combining results that can potentially be $\perp$), and $\oplus$ (for combining proper results):

$$
\begin{array}{ll}
res_\bot \oplus_\bot res'_\bot = & (b_1, con_1) \oplus (b_2, con_2) = \\
\quad \textbf{case } (res_\bot, res'_\bot) \textbf{ of} & \quad (b_1 \wedge b_2, con_1 \cup con_2) \\
\quad\quad (\bot, r) \Rightarrow r & \\
\quad\quad (r, \bot) \Rightarrow r & \\
\quad\quad (r_1, r_2) \Rightarrow r_1 \oplus r_2 &
\end{array}
$$

Note that this semantics will yield a status (*true* or *false*) for each observed event depending on whether an error state has been entered in that specific transition. This status does not reflect whether an error state has been entered *so far* from the beginning of the event stream. This form of non-monotonic result computation allows the result to switch for example from *false* in one step to *true* in the next, and is useful for online monitoring, where it is desirable to know whether the *current* event causes an error. The result across the trace can simply be computed as the conjunction over all emitted status flags. In case a 4-valued logic is desired [9], this is easily calculated on the basis of the contents of the current configuration (*false*: if **error** reached, and if not, *true*: if it contains no states, *possibly false*: if it contains at least one non-final state, and *possibly true*: if it contains only final states, one or more).

Operations Semantics. The LTS denoted by a monitor is defined by the operational semantics presented in Figure 2. The semantics is defined for the kernel language not including (i) **always** states, (ii) transitions at the outermost level, and (iii) inlined states (all states have to be explicitly named).

Rule E (Evaluate) is the top-level rule, and reads as follows. A configuration *con* evolves ($\xrightarrow{e,b}$ below the line) to a configuration con' on observation of an event e, while emitting a status flag b, if ($\overset{e}{\hookrightarrow}$ above the line): con, con, where the second con functions as an iterator, yields the status b and configuration con'.

Rule E-ss$_1$ (Evaluate set of states) defines how the state iterator set is traversed ($\overset{e}{\hookrightarrow}$), here in the situation where the state iterator set has become empty. Rule E-ss$_2$ defines the evaluation in the case where the state iterator is not empty, by selecting a state s, which then is evaluated using $\overset{e}{\longmapsto}$, and then evaluating the remaining states ss recursively with $\overset{e}{\hookrightarrow}$.

Rule E-s$_1$ (Evaluate state) defines the evaluation of a state ($\overset{e}{\longmapsto}$) by evaluating ($\overset{e}{\Longrightarrow}$) its transitions $t.ts$ in the configuration and in the environment $t.env$ associated with the state. Here in the situation where none of the transitions fire, represented above the line by the result of $\overset{e}{\Longrightarrow}$ being the value $\bot$. Rule E-s$_2$ defines the evaluation in the situation where at least one of the transitions fire.

Rule E-ts$_1$ (Evaluate transitions) defines the evaluation ($\overset{e}{\Longrightarrow}$) of a list of transitions in the environment of the current state being evaluated. Here in the situation where this list is empty. In this case $\bot$ is returned to indicate that no transitions fired. Rule E-ts$_2$ defines the evaluation in the case where there is at least one transition t to be evaluated using $\overset{e}{\rightharpoonup}$ to a result, potentially $\bot$, and then evaluating the remaining transitions ts recursively with $\overset{e}{\Longrightarrow}$.

$$\boxed{\text{E}}\ \frac{con, con \overset{e}{\hookrightarrow} b, con'}{con \xrightarrow{e,b} con'}$$

$$\boxed{\text{E-ss}_1}\ \frac{}{con, \{\} \overset{e}{\hookrightarrow} (true, \{\})}$$

$$\boxed{\text{E-ss}_2}\ \frac{\begin{array}{c} con, s \xmapsto{e} res \\ con, ss \overset{e}{\hookrightarrow} res' \end{array}}{con, s \cup ss \overset{e}{\hookrightarrow} res \oplus res'}$$

$$\boxed{\text{E-s}_1}\ \frac{con, s.env, s.ts \overset{e}{\Longrightarrow} \bot}{con, s \xmapsto{e} true, \{s\}}$$

$$\boxed{\text{E-s}_2}\ \frac{con, s.env, s.ts \overset{e}{\Longrightarrow} res}{con, s \xmapsto{e} res}$$

$$\boxed{\text{E-ts}_1}\ \frac{}{con, env, Nil \overset{e}{\Longrightarrow} \bot}$$

$$\boxed{\text{E-ts}_2}\ \frac{\begin{array}{c} con, env, t \overset{e}{\rightharpoonup} res_\bot \\ con, env, ts \overset{e}{\Longrightarrow} res'_\bot \end{array}}{con, env, \langle t \rangle ^\frown ts \overset{e}{\Longrightarrow} res_\bot \oplus_\bot res'_\bot}$$

$$\boxed{\text{E-t}_1}\ \frac{\begin{array}{c} t \textbf{ is } `pat :: cond \to rhs' \\ [\![pat]\!]^P env\ e = \bot \end{array}}{con, env, t \overset{e}{\rightharpoonup} \bot}$$

$$\boxed{\text{E-t}_2}\ \frac{\begin{array}{c} t \textbf{ is } `pat :: cond \to rhs' \\ [\![pat]\!]^P env\ e = env' \\ [\![cond]\!]^C con\ env' = false \end{array}}{con, env, t \overset{e}{\rightharpoonup} \bot}$$

$$\boxed{\text{E-t}_3}\ \frac{\begin{array}{c} t \textbf{ is } `pat :: cond \to rhs' \\ [\![pat]\!]^P env\ e = env' \\ [\![cond]\!]^C con\ env' = true \\ [\![rhs]\!]^R con\ env' = res \end{array}}{con, env, t \overset{e}{\rightharpoonup} res}$$

Fig. 2. Operational semantics of DAUT

Finally, rule E-t$_1$ (Evaluate transition) defines the evaluation ($\xrightarrow{e}$) of a monitor transition t, which has the syntactic format: *pat* :: *cond* $\rightarrow$ *rhs*, in the environment of the current state being evaluated. Recall that a transition consists of a pattern *pat* against which an observed event is matched. If successfully matched, the condition *cond* is evaluated, and if *true*, the right-hand side action *rhs* is executed. Rule E-t$_1$ defines the evaluation in the situation where the pattern *pat* does not match the event, either because the event names differ or because the actual parameters of the event do not match the assignments to the formal parameters defined by *env*. In this case $\perp$ is returned to indicate that no transitions fired. The semantics of patterns is defined by the evaluator $[\![\_]\!]^P$ in Figure 3. Rule E-t$_2$ defines the evaluation in the case where the pattern does match, but where the condition, evaluated by $[\![\_]\!]^C$ in Figure 3, evaluates to *false*. Rule E-t$_3$ defines the evaluation in the case where the pattern matches and the condition evaluates to *true*. In this case the right-hand side *rhs* is evaluated with $[\![\_]\!]^R$ in Figure 4.

Semantic Functions. The semantic functions referenced in Figure 2 are defined in Figures 3 and 4. The semantics of expressions is the obvious one and is not spelled out. The semantics of conditions is also the obvious one, except for the semantics of state predicates of the form $id(exp_1, \ldots, exp_n)$: the expression arguments are evaluated and the result is *true* if and only if the resulting state is contained in the configuration *con*[1]. The semantics of the right-hand side, a comma separated list of actions of type $Action^{**}$, is obtained by evaluating each action to a result, and then 'and' ($\wedge$) the flags together and 'union' ($\cup$) the configurations together. The semantics of an action is a pair consisting of a status flag and a configuration, the flag being *false* if the action is **error**.

4 Optimization

The operational semantics presented in Figure 2 in the previous section is based on iterating through the configuration, a set of states, (rules E-ss$_1$ and E-ss$_2$), state by state, evaluating the event against each state. This is obviously costly. A better approach is to arrange the configuration as an indexed structure which makes it efficient for a given event to extract exactly those states that have transitions labeled with event patterns where the event name is the same, and where the formal parameters are bound to values (in the state's environment *env*) that match those in the corresponding positions in the incoming event. We here ignore "don't care" patterns, which match any event (the actually implemented algorithm deal with these as well). In the following we highlight some of the classes implementing such an optimization in the SCALA programming language. First the top-level *Monitor* class:

[1] The actually implemented semantics is a little more complicated by allowing selected arguments to the state predicate to be the "don't care" value '_', meaning that the search will not care about the values in these positions. However, the automaton concept is meaningful without this additional feature.

```
$[\![\_]\!]^P : Pattern \rightarrow Env \rightarrow Event \rightarrow Env_\perp$
$[\![pat]\!]^P env\ id(v_1, \ldots, v_n) =$
    case $pat$ of
        "_" $\Rightarrow env$ // don't care pattern matches all
        $id(id_1, \ldots, id_n) \Rightarrow$ // event names match
            let $env' = \{id_1 \mapsto v1, \ldots, id_n \mapsto v_n\}$ in
                if $(\forall id \in (dom(env) \cap dom(env')) \bullet env(id) = env'(id)))$
                    then $env \oplus env'$
                    else $\perp$ // bindings do not match
        $id'(\ldots)$ where $id \neq id' \Rightarrow \perp$ // event names do not match

$[\![\_]\!]^C : Cond \rightarrow Config \rightarrow Env \rightarrow \mathbb{B}$
$[\![cond]\!]^C con\ env =$
    case $cond$ of
        ...
        $id(exp_1, \ldots, exp_n) \Rightarrow id([\![exp_1]\!]env, \ldots, [\![exp_n]\!]env)) \in con$

$[\![\_]\!]^E : Exp \rightarrow Env \rightarrow \mathbb{B}$
...
```

Fig. 3. Semantics of patterns, conditions and expressions

```
class Monitor(automaton: Automaton) {
  val config = new Config(automaton)
  ...
  def verify(event: Event) {
    var statesToRem: Set[State] = {}
    var statesToAdd: Set[State] = {}
    for (state ∈ config.getStates(event)) { // efficient search for states
      val (rem, add) = execute(state, event)
      statesToRem ++= rem
      statesToAdd ++= add
    }
    statesToRem foreach config.removeState
    statesToAdd foreach config.addState
  }
}
```

The monitor (parameterized with the abstract syntax tree, *automaton*, representing the monitor) contains a instantiation of the *Configuration* class. The *verify* method is called for each event. It maintains two sets, one containing states to be removed from the configuration as a result of taking transitions, and one for containing states to be added. These sets are used to update the

```
$[\![\_]\!]^R : Action^{**} \rightarrow Config \rightarrow Env \rightarrow Result$
$[\![act_1, \ldots, act_n]\!]^R con\ env =$
    let
        $results = \{[\![act_i]\!] con\ env \mid i \in 1..n\}$
        $status = \bigwedge\{b \mid (b, con') \in results\}$
        $con'' = \bigcup\{con' \mid (b, con') \in results\}$
    in
        $(status, con'')$

$[\![\_]\!]^A : Action \rightarrow Config \rightarrow Env \rightarrow Result$
$[\![act]\!]^A con\ env =$
    case $act$ of
        ok $\Rightarrow (true, \{\})$
        error $\Rightarrow (false, \{\})$
        $id(exp_1, \ldots, exp_n) \Rightarrow (true, \{id([\![exp_1]\!]env, \ldots, [\![exp_n]\!]env)\})$
        if $(cond)$ then $act_1$ else $act_2$ $\Rightarrow$
            if $([\![cond]\!]con\ env)$ then $[\![act_1]\!]con\ env$ else $[\![act_2]\!]con\ env$
```

Fig. 4. Semantics of transition right-hand sides

configuration at the end of the method. The essential part of this method is the expression: *config.getStates(event)*, which extracts only the relevant states for a given event.

The *Configuration* class is defined next. The core idea is to maintain two kinds of nodes: *state nodes* and *event nodes*. There is one state node for each named state. It contains at any point in time an index of all the states with that name, only distinguished by their parameters. Likewise, there is one event node for each transition, representing the event pattern on that transition. The event node is linked to the source state of the transition. The state nodes and event nodes are mapped to by their names. Since an event name can occur on several transitions, an event name is mapped to a list of event nodes:

```
class Config(automaton: Automaton) {
  var stateNodes: Map[String, StateNode] = Map()
  var eventNodes: Map[String, List[EventNode]] = Map()
   ...
  def getStates(event: Event): Set[State] = {
    val (eventName, values) = event
    var result: Set[State] = Set()
    eventNodes.get(eventName) match {
      case None ⇒
      case Some(eventNodeList) ⇒
        for (eventNode ∈ eventNodeList) {
          result ++= eventNode.getRelevantStates(event)
        }
      }
      result
    }
  }
```

The method *getStates* returns the set of states relevant for a given event. It does this by first looking up all the event nodes for that event (those with the same name), each corresponding to a particular transition, and for each of these it retrieves the relevant states in the corresponding state node. The details of how this works is given by the classes *EventNode* and *StateNode*, where sets and maps are mutable (updated point wise for efficiency reasons). The class *EventNode* is as follows:

```
case class EventNode(stateNode: StateNode,
  eventIds: List[Int], stateIds: List[String]) {
   ...
  def getRelevantStates(event: Event): Set[State] = {
    val (_, values) = event
    stateNode.get(
      stateIds,
      for (eventId ∈ eventIds) yield values(eventId)
    )
  }
}
```

An event node contains a reference to the state node it is connected to (the source state of the transition the event pattern occurs on), a list of parameter positions in the event that are relevant for the search of relevant states, and a list of the formal parameter names in the associated state these parameter positions correspond to. To calculate the states relevant for an event, the state node's *get* method is called with two arguments: the list of formal state parameters that are relevant, and the list of values they have in the observed event. The state node is as follows:

```
case class StateNode(stateName: String, paramIdList: List[String]) {
  var index: Map[List[String], Map[List[Value], Set[State]]] = Map()
   ...
  def get(paramIdList: List[String],valueList: List[Value]): Set[State] =
  {
    index(paramIdList).get(valueList) match {
      case None ⇒ emptySet
      case Some(stateSet) ⇒ stateSet
    }
  }
}
```

Listing 4. A monitor with a cancel option

```
monitor R3 {
  grant(t, r) → Granted(t,r)

  hot Granted(t,r) {
    release(t,r) → ok
    cancel(r) → ok
  }
}
```

A state node defines the name of the state, as well as its parameter identifier list (formal parameters). It contains an index, which maps a projection of the parameter identifiers to yet a map, which maps lists of values for these parameters to states which bind exactly those values to those parameters. A similar *put* method is defined, which inserts a state in the appropriate slot.

As an example, consider the monitor in Listing 4, where a depletable resource (can be assigned simultaneously to more than one task) either can get released by the task that it was granted to, or it can be canceled for all tasks that currently hold it. Suppose we observe the events $\langle grant(t_1, a), grant(t_2, a)\rangle$. Then the index for the state node for *Granted* will look as follows:

$$\begin{array}{lll} \langle t, r\rangle & \mapsto & [\ \langle t_1, a\rangle \mapsto \{Granted(t_1, a)\},\ \langle t_2, a\rangle \mapsto \{Granted(t_2, a)\}\ ] \\ \langle r\rangle & \mapsto & [\ \langle a\rangle \mapsto \{Granted(t_1, a), Granted(t_2, a)\}\ ] \end{array}$$

5 Internal DSL

The internal DSL, DAUT^{int}, is defined as an API in SCALA. SCALA offers various features which can can make an API look and feel like a DSL. These include implicit functions, possibility to omit dots and parentheses in calls of methods on objects (although not used here), partial functions, pattern matching, and case classes. DAUT^{int} is a variation of TRACECONTRACT, presented in [5], which explains in more detail how to use SCALA for defining a domain specific language for monitoring. TRACECONTRACT is a larger DSL, also including an embedding of linear temporal logic. However, it does in its pure form not support specification of past time properties (additional rule-based constructs had to be added to support this). DAUT^{int} is much simpler, just focusing on data automata, and it supports specification of past time properties by allowing transition conditions to refer to other parameterized states. This is achieved by defining states as **case** classes. A main advantage of an internal DSL is the ability to mix the DSL with code. Although not shown here, monitors can freely mix DSL constructs and programming constructs, such as variable declarations and assignment statements. For example, the right-hand side of a transition can include SCALA statements.

Listing 5. Events (DAUT^{int})

```
trait Event
case class grant(task: String, resource: String) extends Event
case class release(task: String, resource: String) extends Event
```

The complete implementation of DAUT^{int} is shown in Appendix A. As shown in Section 6, this simple DSL is surprisingly efficient compared to many other systems (except for MOP), which is interesting considering that it consists of very few lines of code. We shall not here explain the details, and refer to [5] for the general principles of implementing a similar DSL. Instead we shall illustrate what the DAUT monitors presented in Section 3 look like in DAUT^{int}. First we need to define the events of interest, see Listing 5. This is done by introducing the trait (similar to an abstract class) of events *Event* and then defining each type of event as a case class subclassing *Event*. In contrast to normal classes, case classes allow pattern matching over objects of the class, including its parameters. The monitors in listings 2 and 3 can be programmed as shown in Listing 6.

Note the similarity with the corresponding DAUT monitors. A monitor extends the *Monitor* class, which is parameterized with the event type. The method *whenever* takes a partial function as argument and creates an initial **always** state from it. A partial function can in SCALA be defined with a sequence of **case** statements using pattern matching over the events, defining the domain of the partial function. A state is modeled as a class that subclasses one of the pre-defined classes: *state*, *hot*, or *always*, defining respectively normal final states, non-final states, and final states with self-loops. The transitions in a state are declared with the *when* method which, just as the *whenever* method, takes a partial function representing the transitions as argument. Note that in order to enforce a pattern to match on values bound to an identifier, the identifier has to be quoted, as in ‘t‘. Finally, DAUT^{int} allows to combine monitors in a hierarchical manner, for the purpose of grouping monitors together. A monitor can be applied as shown in Listing 7, creating an instance and subsequently submitting events to it.

6 Evaluation

This section describes the benchmarking performed to evaluate DAUT, the abstract operational semantics DAUT^{sos}, and the internal DSL, DAUT^{int}. The systems are evaluated against seven other RV systems, also evaluated in [15], which also explains the evaluation setup in details. The experiments focus on analysis of logs (offline analysis), since this has been the focus of our application of RV. The evaluation was carried out on an Apple Mac Pro, 2×2.93 GHz 6-Core Intel Xeon, 32GB of memory, running Mac OS X Lion 10.7.5. Applications were run

Listing 6. Monitors (DAUT^{int})

```
class R1R2 extends Monitor[Event] {
  whenever {
    case grant(t, r) ⇒ Granted(t, r)
    case release(t, r) if !Granted(t, r) ⇒ error
  }

  case class Granted(t: String, r: String) extends hot {
    when {
      case release(‘t‘, ‘r‘) ⇒ ok
      case grant(_, ‘r‘) ⇒ error
    }
  }
}

class R1 extends Monitor[Event] {
  whenever {
    case grant(t, r) ⇒ hot {
      case release(‘t‘, ‘r‘) ⇒ ok
      case grant(_, ‘r‘) ⇒ error
    }
  }
}
```

Listing 7. Applying a monitor (DAUT^{int})

```
object Main {
  def main(args: Array[String]) {
    val obs = new R1R2

    obs.verify(grant("t1", "A"))
    obs.verify(release("t1", "A"))

    obs.end()
  }
}
```

in Eclipse JUNO 4.2.2, running Scala IDE version 3.0.0/2.10 and Java 1.6.0. The systems compared are explained in [15]. All monitors check requirements R_1 and R_2 (page 257), formalized in Daut in Listing 2 and in Daut^{int} in Listing 6 (first monitor). Logs can abstractly be seen as sequences of events $\mathit{grant}(t,r)$ and $\mathit{release}(t,r)$, where t and r are integer values. The logs are represented as CSV files, and parsed with a CSV-parsing script.

Table 1. Results of tests 1-7. For each test is shown the *memory* of the test, length of the trace, and time taken to parse the log (subtracted in the following numbers). For each tool two numbers are provided - above line: number of events processed by the monitor per millisecond, and below line: time consumed monitoring (minutes:seconds:milliseconds, with minutes and seconds left out if 0). DNF stands for 'Did Not Finish'.

trace nr.	1	2	3	4	5	6	7
memory	1	1	5	30	100	500	5000
length	30,933	2,000,002	2,100,010	2,000,060	2,000,200	2,001,000	1,010,000
parsing	3 sec	45 sec	47 sec	46 sec	46 sec	46 sec	24 sec
LogFire	26 / 1:190	42 / 47:900	41 / 50:996	34 / 58:391	23 / 1:27:488	8 / 3:55:696	1 / 15:54:769
Rete/UL	38 / 816	109 / 18:428	75 / 28:141	41 / 48:524	14 / 2:26:983	4 / 8:25:867	0.4 / 43:33:366
Drools	10 / 3:97	8 / 4:1:758	9 / 3:47:535	9 / 3:34:648	8 / 4:14:497	7 / 4:36:608	3 / 5:4:505
Ruler	95 / 326	138 / 14:441	78 / 27:77	8 / 4:5:593	0.8 / 41:39:750	0.034 / 977:20:636	DNF
LogScope	17 / 1:842	15 / 2:11:908	7 / 4:54:605	2 / 21:42:389	0.4 / 76:17:341	0.09 / 369:25:312	0.01 / 2074:43:470
TraceContract	48 / 645	69 / 28:851	37 / 57:428	6 / 5:58:497	0.9 / 36:29:594	0.036 / 919:5:134	DNF
Daut	49 / 631	84 / 23:847	86 / 24:338	89 / 22:432	90 / 22:298	86 / 23:287	80 / 12:612
Daut^{sos}	102 / 302	192 / 10:435	79 / 26:438	24 / 1:22:727	8 / 4:19:697	2 / 16:27:990	0.18 / 92:2:26
Daut^{int}	233 / 133	1715 / 1:166	770 / 2:729	373 / 5:368	195 / 10:236	54 / 36:929	5 / 3:6:560
Mop	595 / 52	1381 / 1:448	1559 / 347	1341 / 1:491	7143 / 280	7096 / 282	847 / 1:193

The experiment consists of analyzing seven different logs: one log, numbered 1, generated from the Mars Curiosity rover during 99 (Mars) days of operation on Mars, together with six artificially generated logs, numbered 2-7, that are supposed to stress test the algorithms for their ability to handle particular situations requiring fast indexing. The MSL log contains a little over 2.4 million events, of which 30.933 are relevant *grant* and *release* events, which are extracted before analysis. The shape of this log is a sequence of paired *grant* and *release* events, where a resource is released in the step immediately following the grant event (after all other events have been filtered out). In this case we say that the required *memory* is 1: only one (*task*, *resource*) association needs to be remembered at any point in time. In this sense there is no need for indexing since only one resource is held at any time. This might be a very realistic scenario in many cases. The artificially generated logs experiment with various levels of *memory* amongst the values: $\{1, 5, 30, 100, 500, 5000\}$. As an example, a memory value of 500 means that the log contains 500 $grant(t, r)$ events for all different values of (t, r), before any resources are released, resulting a memory of size 500, which then has to be indexed. The results are shown in Table 1.

The table shows that Mop outperforms all other systems by orders of magnitude. This fundamentally illustrates that the indexing approach used, although leading to limited expressiveness, has major advantages when it comes to efficiency. A more surprising result, however, is that the internal DSL Daut^{int} outperforms all other tools, except Mop, for lower memory values. Furthermore, as a positive result, the optimized Daut presented in this paper performs better than the other systems (again except Mop) for high memory values.

7 Conclusion

We have presented data automata, their syntax, semantics and efficient implementation. We consider data automata as providing a natural solution to the monitoring problem. The formalism and indexing algorithm have been motivated based on our experiences with rule-based systems, hence exploring the space between standard propositional automata and fully general rule-based systems. The algorithm is much less complex than the Rete algorithm, often used in rule-based systems, and appears to be more efficient. However, the implementation is not as efficient as the state-of-the-art RV system Mop. On the other hand, the notation is more expressive. We have shown an implementation in Scala of an internal DSL which models data automata, but with the additional advantage of providing all of Scala's features. The implementation is very simple, but moderately competitive wrt. efficiency.

References

1. Allan, C., Avgustinov, P., Christensen, A.S., Hendren, L., Kuzins, S., Lhoták, O., de Moor, O., Sereni, D., Sittamplan, G., Tibble, J.: Adding trace matching with free variables to AspectJ. In: OOPSLA 2005. ACM Press (2005)
2. Barringer, H., Falcone, Y., Havelund, K., Reger, G., Rydeheard, D.: Quantified Event Automata: Towards Expressive and Efficient Runtime Monitors. In: Giannakopoulou, D., Méry, D. (eds.) FM 2012. LNCS, vol. 7436, pp. 68–84. Springer, Heidelberg (2012)
3. Barringer, H., Goldberg, A., Havelund, K., Sen, K.: Rule-based runtime verification. In: Steffen, B., Levi, G. (eds.) VMCAI 2004. LNCS, vol. 2937, pp. 44–57. Springer, Heidelberg (2004)
4. Barringer, H., Groce, A., Havelund, K., Smith, M.: Formal analysis of log files. J. of Aerospace Computing, Information, and Communication 7(11), 365–390 (2010)
5. Barringer, H., Havelund, K.: TraceContract: A Scala DSL for trace analysis. In: Butler, M., Schulte, W. (eds.) FM 2011. LNCS, vol. 6664, pp. 57–72. Springer, Heidelberg (2011)

6. Barringer, H., Rydeheard, D.E., Havelund, K.: Rule systems for run-time monitoring: from Eagle to RuleR. J. Log. Comput. 20(3), 675–706 (2010)
7. Basin, D., Klaedtke, F., Müller, S.: Policy monitoring in first-order temporal logic. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 1–18. Springer, Heidelberg (2010)
8. Bauer, A., Küster, J.-C., Vegliach, G.: From propositional to first-order monitoring. In: Legay, A., Bensalem, S. (eds.) RV 2013. LNCS, vol. 8174, pp. 59–75. Springer, Heidelberg (2013)
9. Bauer, A., Leucker, M., Schallhart, C.: The good, the bad, and the ugly, but how ugly is ugly? In: Sokolsky, O., Taşıran, S. (eds.) RV 2007. LNCS, vol. 4839, pp. 126–138. Springer, Heidelberg (2007)
10. Bodden, E.: MOPBox: A library approach to runtime verification. In: Khurshid, S., Sen, K. (eds.) RV 2011. LNCS, vol. 7186, pp. 365–369. Springer, Heidelberg (2012)
11. Decker, N., Leucker, M., Thoma, D.: Monitoring modulo theories. In: Ábrahám, E., Havelund, K. (eds.) TACAS 2014. LNCS, vol. 8413, pp. 341–356. Springer, Heidelberg (2014)
12. Forgy, C.: Rete: A fast algorithm for the many pattern/many object pattern match problem. Artificial Intelligence 19, 17–37 (1982)
13. Goubault-Larrecq, J., Olivain, J.: A smell of ORCHIDS. In: Leucker, M. (ed.) RV 2008. LNCS, vol. 5289, pp. 1–20. Springer, Heidelberg (2008)
14. Hallé, S., Villemaire, R.: Runtime enforcement of web service message contracts with data. IEEE Transactions on Services Computing 5(2), 192–206 (2012)
15. Havelund, K.: Rule-based runtime verification revisited. Software Tools for Technology Transfer (STTT) (April 2014) (published online)
16. Meredith, P., Jin, D., Griffith, D., Chen, F., Roşu, G.: An overview of the MOP runtime verification framework. Software Tools for Technology Transfer (STTT) 14(3), 249–289 (2012)
17. Stolz, V.: Temporal assertions with parametrised propositions. In: Sokolsky, O., Taşıran, S. (eds.) RV 2007. LNCS, vol. 4839, pp. 176–187. Springer, Heidelberg (2007)
18. Stolz, V., Bodden, E.: Temporal assertions using AspectJ. In: Proc. of the 5th Int. Workshop on Runtime Verification (RV 2005). ENTCS, vol. 144(4), pp. 109–124. Elsevier (2006)

A The Internal Scala DSL DAUT^{int}

```
class Monitor[E <: AnyRef] {
  val monitorName =
    this.getClass().getSimpleName()

  var monitors: List[Monitor[E]] = List()
  var states: Set[state] = Set()

  var statesToAdd: Set[state] = Set()
  var statesToRemove: Set[state] = Set()

  def monitor(monitors: Monitor[E]*) {
    this.monitors ++= monitors
  }

  type Transitions =
    PartialFunction[E, Set[state]]

  def noTransitions: Transitions =
  {
    case _ if false ⇒ null
  }

  class state {
    var transitions: Transitions =
      noTransitions

    def when(ts: Transitions) {
      this.transitions = ts
    }

    def apply(event: E): Option[Set[state]] =
      if (transitions.isDefinedAt(event))
        Some(transitions(event)) else None
  }

  class always extends state
  class hot extends state
  case object error extends state
  case object ok extends state

  def stateExists(
    pred: PartialFunction[state, Boolean]):
      Boolean =
  {
    states exists (pred orElse {
      case _ ⇒ false })
  }

  def state(ts: Transitions): state =
  {
    val e = new state
    e.when(ts)
    e
  }

  def always(ts: Transitions): state =
  {
    val e = new always
    e.when(ts)
    e
  }

  def hot(ts: Transitions): state =
  {
    val e = new hot
    e.when(ts)
    e
  }

  def error(msg: String): state =
  {
    println("\n*** " + msg + "\n")
    error
  }

  def whenever(ts: Transitions) {
    states += always(ts)
  }


  implicit def stateToBoolean(s: state): Boolean =
    states contains s

  implicit def unitToSet(u: Unit): Set[state] =
    Set(ok)

  implicit def stateToSet(s: state): Set[state] =
    Set(s)

  implicit def statePairToSet(
    ss: (state, state)): Set[state] =
      Set(ss._1, ss._2)

  implicit def stateTripleToSet(
    ss: (state, state, state)): Set[state] =
      Set(ss._1, ss._2, ss._3)


  def verify(event: E) {
    for (s ∈ states) {
      s(event) match {
        case None ⇒
        case Some(stateSet) ⇒
          if (stateSet contains error) {
            println("\n*** error!\n")
          } else {
            for (state ∈ stateSet) {
              if (state != ok) {
                statesToAdd += state
              }
            }
          }
          if (!s.isInstanceOf[always]) {
            statesToRemove += s
          }
      }
    }
    states --= statesToRemove
    states ++= statesToAdd
    statesToAdd = Set()
    statesToRemove = Set()
    for (monitor ∈ monitors) {
      monitor.verify(event)
    }
  }

  def end() {
    val hotStates =
      states filter (_.isInstanceOf[hot])
    if (!hotStates.isEmpty) {
      println("*** hot states in " + monitorName)
      hotStates foreach println
    }
    for (monitor ∈ monitors) {
      monitor.end()
    }
  }
}
```

Risk-Based Testing
(Track Introduction)

Michael Felderer[1], Marc-Florian Wendland[2], and Ina Schieferdecker[2]

[1] University of Innsbruck, Innsbruck, Austria
michael.felderer@uibk.ac.at
[2] Fraunhofer Institute FOKUS, Berlin, Germany
{marc-florian.wendland,ina.schieferdecker}@fokus.fraunhofer.de

1 Motivation and Goals

In many development projects, testing has to be done under severe pressure due to limited resources, a challenging time schedule, and the demand to guarantee security and safety of the released software system. Risk-based testing, which utilizes identified risks of a software system for testing purposes, has a high potential to improve testing in this context. It optimizes the allocation of resources and time, is a means for mitigating risks, helps to early identify critical areas, and provides decision support for the management [1, 2]. Risk-based testing is a type of software testing that explicitly considers risks of the software system as the guiding factor to solve decision problems in all phases of the test process, i.e., test planning, design, implementation, execution and evaluation [3–5]. It is based on the intuitive idea to focus testing activities on those areas that trigger the most critical situations for a software system [6]. The precise understanding of risks as well as their focused treatment by risk-based testing has become one of the cornerstones for critical decisions within complex software development projects and recently gained much attention [7]. Lately, the international standard ISO/IEC/IEEE 29119 Software Testing [8] on testing techniques, processes and documentation even explicitly considers risks as an integral part of the test planning process. As a result, several risk-based testing approaches (e.g., [9] or [10]) and empirical studies (e.g., [11] or [12]) have recently been provided to address increased practical need in this area, but further research is still inevitable.

This special track on risk-based testing serves as a platform for researchers and practitioners to present approaches, results, experiences and advances in risk-based testing. Its goal was to bring together researchers and practitioners working in the area of risk-based testing to discuss actual challenges and solutions to them. For this purpose, we invited leading researchers and practitioners to present their solutions to tackle actual challenges of risk-based testing. The invited format ensured broad coverage of this important topic. All contributed papers represent systematic rather than ad-hoc proposals which makes them interesting for a wide audience. Together, the papers in this track, which are summarized in the next section, provide a comprehensive and up-to-date overview of the communitys response to challenges of risk-based testing.

2 Contributions

The special track comprises six contributed papers summarized in the following paragraphs.

Seehusen [13] presents a technique for risk-based test procedure identification, prioritization, and selection. The technique takes a risk model in the form of a risk graph as input, and produces a list of prioritized selected test procedures as output. The technique is generic as it can be used with many existing risk documentation languages and many kinds of likelihood and risk types. In the paper, the technique is demonstrated on the CORAS threat diagram language [14].

Felderer et al. [15] present a framework for integrating risk assessment, i.e., risk identification, analysis and evaluation, into an established test process. Their framework contains a risk assessment model which configures the test process. This model and its artifacts therefore determine the overall risk-based test process and are the main component of their risk assessment framework for testing purposes. The risk assessment model defines the test scope, the risk identification method, a risk model, and the tooling for risk assessment. It is derived on the basis of best practices extracted from published risk-based testing approaches and applied to an industrial test process.

Yahav et al. [16] address the quality risk of open source software components. For this purpose, Yahav et al. predict occurrence of bugs in these components using communication and community data, i.e., data on email communication traffic and social network dynamics on the basis of regression models. The information on predicted bugs is then intended to be used to allocate test efforts. The approach is illustrated with data from four open source projects.

Grossmann et al. [17] present an approach called Risk-Based Security Testing that combines risk analysis and risk-based test design activities based on formalized security test patterns. The involved security test patterns are formalized by using a minimal test design strategies language framework which is represented as a UML profile. Such a (semi-)formal security test pattern is then used as the input for a test generator accompanied by the test design model out of which the test cases are generated. The approach is based on the CORAS method [14] for risk analysis activities. Finally, a tool prototype is presented which shows how to combine the CORAS-based risk analysis with pattern-based test generation.

Botella et al. [18] describe an approach to security testing called Risk-Based Vulnarability Testing, which is guided by risk assessment and coverage to perform and automate vulnerability testing for web applications. Risk-Based Vulnerability testing adapts model-based testing techniques using a pattern-based approach for the generation of test cases according to previously identified risks and criticalities. For risk identification and analysis, the CORAS method [14] is utilized. The integration of information from risk analysis activities with the model-based test generation approach is realized by a test purpose language. It is used to formalize security test patterns in order to make them usable for test generators. Risk-Based Vulnerability Testing is applied to security testing of a web application.

References

1. Felderer, M., Haisjackl, C., Breu, R., Motz, J.: Integrating manual and automatic risk assessment for risk-based testing. In: Biffl, S., Winkler, D., Bergsmann, J. (eds.) SWQD 2012. LNBIP, vol. 94, pp. 159–180. Springer, Heidelberg (2012)
2. Felderer, M., Ramler, R.: Experiences and challenges of introducing risk-based testing in an industrial project. In: Winkler, D., Biffl, S., Bergsmann, J. (eds.) SWQD 2013. LNBIP, vol. 133, pp. 10–29. Springer, Heidelberg (2013)
3. Gerrard, P., Thompson, N.: Risk-based e-business testing. Artech House Publishers (2002)
4. Schieferdecker, I., Grossmann, J., Schneider, M.: Model-based security testing. In: Proceedings 7th Workshop on Model-Based Testing (2012)
5. Felderer, M., Ramler, R.: Integrating risk-based testing in industrial test processes. Software Quality Journal 22(3), 543–575 (2014)
6. Wendland, M.F., Kranz, M., Schieferdecker, I.: A systematic approach to risk-based testing using risk-annotated requirements models. In: ICSEA 2012, The Seventh International Conference on Software Engineering Advances, pp. 636–642 (2012)
7. Felderer, M., Schieferdecker, I.: A taxonomy of risk-based testing. STTT (2014), doi:10.1007/s10009-014-0332-3
8. ISO: ISO/IEC/IEEE 29119 Software Testing (2013), `http://softwaretestingstandard.org/` (accessed: August 12, 2014)
9. Neubauer, J., Windmüller, S., Steffen, B.: Risk-based testing via active continuous quality control. STTT (2014), doi:10.1007/s10009-014-0321-6
10. Carrozza, G., Pietrantuono, R., Russo, S.: Dynamic test planning: a study into an industrial context. STTT (2014), doi:10.1007/s10009-014-0319-0
11. Felderer, M., Ramler, R.: A multiple case study on risk-based testing in industry. STTT (2014), doi:10.1007/s10009-014-0328-z
12. Erdogan, G., Li, Y., Runde, R.K., Seehusen, F., Stølen, K.: Approaches for the combined use of risk analysis and testing: A systematic literature review. STTT (2014), doi:10.1007/s10009-014-0330-5
13. Seehusen, F.: A technique for risk-based test procedure identification, prioritization and selection. In: Margaria, T., Steffen, B. (eds.) ISoLA 2014, Part II. LNCS, vol. 8803, pp. 277–291. Springer, Heidelberg (2014)
14. Lund, M.S., Solhaug, B., Stolen, K.: Model-driven Risk Analysis. Springer (2011)
15. Felderer, M., Haisjackl, C., Pekar, V., Breu, R.: A risk assessment framework for software testing. In: Margaria, T., Steffen, B. (eds.) ISoLA 2014, Part II. LNCS, vol. 8803, pp. 292–308. Springer, Heidelberg (2014)
16. Yahav, I., Kenett, R.S., Bai, X.: Data driven testing of open source software. In: Margaria, T., Steffen, B. (eds.) ISoLA 2014, Part II. LNCS, vol. 8803, pp. 309–321. Springer, Heidelberg (2014)
17. Großmann, J., Schneider, M., Viehmann, J., Wendland, M.-F.: Combining risk analysis and security testing. In: Margaria, T., Steffen, B. (eds.) ISoLA 2014, Part II. LNCS, vol. 8803, pp. 322–336. Springer, Heidelberg (2014)
18. Botella, J., Legeard, B., Peureux, F., Vernotte, A.: Risk-based vulnerability testing using security test patterns. In: Margaria, T., Steffen, B. (eds.) ISoLA 2014, Part II. LNCS, vol. 8803, pp. 337–352. Springer, Heidelberg (2014)

A Technique for Risk-Based Test Procedure Identification, Prioritization and Selection

Fredrik Seehusen

Department for Networked Systems and Services, SINTEF ICT
PO Box 124 Blindern, N-0314 Oslo, Norway
fredrik.seehusen@sintef.no

Abstract. We present a technique for risk-based test procedure identification, prioritization, and selection. The technique takes a risk model in the form of a risk graph as input, and produces a list of prioritized selected test procedures as output. The technique is general in the sense that it can be used with many existing risk documentation languages and many kinds of likelihood and risk types.

Keywords: Risk assessment, testing, security, risk-based testing.

1 Introduction

Risk-based testing is an approach in which risk assessment results are used to guide the testing process. Most risk-based approaches can be classified into one of two categories: (1) approaches that use risk to *prioritize* the *parts* of the system under test where the testing should be focused, and (2) approaches that use risk to *identify* potential test procedures.

Our contribution in this paper is a technique which belongs in the second category. The technique is unique in that risk assessment results are not only used to identify test procedures, but also to prioritize and select the test procedures that should be further refined into concrete test procedures and test cases. There are no other techniques to risk-based testing that we are aware of that address both of these issues.

In this paper, we assume that risk assessment results are documented in the form of a so-called risk graph [3]. A risk graph can be seen as an abstraction of many risk assessment languages. The main research questions addressed in this paper are:

- How can risk graphs be used as a basis for test procedure identification, and what is the best way of doing it?
- What estimates are needed in order to prioritize test procedures identified in a risk graph, and how can these be used to automatically calculate a priority?
- What estimates are need in order to select the test procedures that will be used as a starting point for further testing, and how can we calculate an optimal test procedure selection?

A risk graph can be seen as a set of statements about the world. In the paper, we argue that testing a risk graph corresponds to checking the degree to which its statements are correct. Furthermore, we argue that prioritization should be based on the notions of *severity* and *confidence*. Put simply, severity is an estimate of the impact that a statement of a risk graph has on the risk values, and confidence is an estimate of how confident we are about the correctness of a statement. Finally, we argue that in order to do test procedure selection, we must in addition take into account an estimate of the *effort* it would take to refine and implement the test procedures.

This paper is structured as follows: In Sect. 2, we define risk graphs precisely. In Sect.3, we discuss how to use risk graphs as a basis for test procedure identification. In Sect. 4 and Sect. 5 we present our technique for test procedure prioritization and selection, respectively. In Sect. 6 we give an example for how to apply the technique to a specific risk assessment language. Finally, in Sect. 7 we discuss related work, and in Sect. 8 we provide conclusions and discuss future work.

2 Risk Graphs

A risk graph is a common abstraction of many existing risk modeling languages such as fault trees [10], event trees [9], attack trees [19], Bayesian networks [5], and CORAS threat diagrams [13]. Risk graphs are used as language for structuring events leading to incidents and to estimate the likelihood of incidents [3]. A risk graph consists of a set of nodes and a set of edges between the nodes. Both the nodes and the edges may be assigned likelihood values. The nodes typically represent occurrences of events, and the likelihood value of a node specifies how likely it is that its associated event will occur. Edges represent causal relationships between nodes. Likelihood values of edges should be understood as conditional likelihood values. In risk graphs, nodes may also be assigned consequence values, and the risk value of a node is a function of its likelihood and consequence value.

In the following we first, in Sect. 2.1 define likelihood graphs (graphs with likelihood values). Then we define risk graphs (in Sect. 2.2) in terms of likelihood graphs.

2.1 Likelihood Graphs

In this section, we define likelihood graphs as well as some operations on likelihood graphs which will be needed later for defining the test procedure prioritization technique.

A likelihood graph is a directed acyclic graph whose nodes and edges may be annotated by likelihood values. There are many ways of specifying likelihoods (e.g. as probabilities, frequencies, intervals of these, or probability distributions). In order to define a technique which is applicable regardless of the kind of likelihood values we use, we will parameterize likelihood graphs by the notion of a likelihood structure.

Definition 1. *(Likelihood Structure) A likelihood structure λ is a tuple $(L, \oplus, \otimes, \overline{1}, \overline{0})$ consisting of*

- *a set L of likelihood values;*
- *two elements $\overline{1} \in L$ and $\overline{0} \in L$ known as the maximum and minimum likelihood values of λ, respectively;*
- *a binary operator $\oplus$ on L, known as the or-operator of λ;*
- *a binary operator $\otimes$ on L, known as the and-operator of λ.*

We denote by $\lambda.L$, $\lambda.\overline{0}$, $\lambda.\overline{1}$, $\lambda.\oplus$, and $\lambda.\otimes$, the likelihood values, the minimum value, the maximum value, the or-operator, or the and-operator of λ, respectively. We sometimes drop the λ. prefix when λ is clear from the context.

Example 1. To express likelihoods in terms of, say, mutually exclusive probabilities, we have to instantiate the elements of the likelihood structure as follows

- the set of likelihood values is the set of all real numbers between 0 and 1;
- the maximum value is 1 and the minimum value is 0;
- the or-operator is defined by addition, i.e. $l \oplus l' \triangleq l + l'$;
- the and-operator is defined by multiplication i.e. $l \otimes l' \triangleq l * l'$.

We will call this likelihood structure the mutually exclusive probability structure, and denote it by **p** for reference in later examples.

Having defined likelihood structures, we are now ready to define likelihood graphs.

Definition 2. *(Likelihood Graph) A likelihood graph G over a likelihood structure λ is a tuple (Q, E, l) consisting of*

- *a set of nodes Q;*
- *a set of edges $E \subseteq Q \times Q$;*
- *a partial function $l \in Q \cup E \rightarrow \lambda.L$ assigning likelihood values to nodes and edges.*

We denote by $G.Q$, $G.E$, and $G.l$, the nodes, edges, and likelihood assignment function of G. We sometimes just write Q, E, or l if G is clear from the context. We require that all likelihood graphs be acyclic.

Example 2. Fig. 1 shows an example of a likelihood graph. Here all edges have been assigned probability likelihood values. The nodes are not assigned likelihood values, they are labeled by S_1 to S_6, but this is for reference purposes only. The graph shown in Fig. 1, contains some value annotations that are not part of the likelihood graph (the values *Low*, *Medium*, and *High*, and the values 2 and 5 on node S_5 and S_6). Ignore this for now, it will be explained later.

If $e = (p, q)$ is an edge, then the source and target nodes of the edge are defined by $src(e) \triangleq p$ and $tar(e) \triangleq q$. If G is a likelihood graph and p is a node in G, then we denote by $src(G, p)$, all the edges in G that have p as target, i.e.,

$$src(G, p) \triangleq \{e \in G.E \mid tar(e) = p\}$$

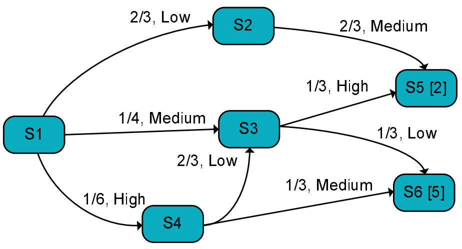


Fig. 1. Example of a risk graph with confidence values on edges

To specify the test procedure prioritization function later, we need to be able to calculate the likelihood values of nodes in a graph based on the likelihood values of the edges. This is defined in the following.

Definition 3. *(Calculated likelihood of a node) Let G be a likelihood graph over a likelihood structure λ where $\lambda.\oplus$ is commutative. Then the likelihood value of a node p in G, written $l[G,p]$, as calculated from the edges in G, is defined by*

$$l[G,p] \triangleq \begin{cases} \bigoplus_{e \in src(G,p)} l(e) \otimes l[G, src(e)] & \text{if } src(G,p) \neq \emptyset \\ \overline{1} & \text{if } src(G,p) = \emptyset \end{cases}$$

Note that the order in which the likelihood values are summed should not matter. Therefore we have required that $\oplus$ must be commutative.

Example 3. Assuming the likelihood structure **p** defined in Example 1, we can calculate all the likelihood values of the nodes of the likelihood graph in Fig. 1 based on the likelihood values of the edges. We then get: $S_1 = 1$, $S_2 = \frac{2}{3}$, $S_3 = \frac{13}{36}$, $S_4 = \frac{1}{4}$, $S_5 = \frac{61}{108}$, and $S_6 = \frac{19}{108}$.

Before we continue, we define a helper function which will be needed later.

Definition 4. *(Weight replacement) If e is an edge in G and l is a likelihood value of the likelihood structure of G, then the likelihood graph obtained by replacing the likelihood value of e by l is denoted by $we(G,e,l)$.*

2.2 Risk Graphs

In this section, we define the notion of a risk graph, which is basically just a likelihood graph whose nodes may be assigned consequence values in addition to likelihood values. Risk values may then be calculated as a function of likelihood and consequence values. Similar to what we did for likelihood graphs, we parameterize risk graphs with the notion of a risk structure.

Definition 5. *(Risk Structure) A risk structure ρ is a tuple $(\lambda, C, R, \odot, rv)$ consisting of*

- *a likelihood structure λ;*
- *a set of consequence values of C and a set of risk values R;*
- *a risk difference operator $\odot \in R \times R \rightarrow \mathbb{R}$ that takes two risk values, and yields the difference, expressed as a real number, between the risks;*
- *a risk value function $rv \in \lambda.L \times C \rightarrow R$ mapping likelihood values of λ and consequence values into risk values.*

We are now ready to define risk graphs.

Definition 6. *(Risk graph) A risk graph G over a risk structure ρ, is a tuple (Q, E, l, c), consisting of*

- *a likelihood graph (Q, E, l) over $\rho.\lambda$;*
- *a partial function $c \in Q \rightarrow \rho.C$ assigning consequence values of ρ to nodes.*

If G is a risk graph, then we denote by $G.Q_c$, or just Q_c if G is clear from the context, the set of all nodes in G that have a consequence value assigned to it.

In a given risk graph, we refer to all nodes that have a likelihood and a consequence value as risk nodes, or just risks for short.

Example 4. A typical risk structure in our experience uses real values as consequences and defines the risk value as the product of a likelihood and a consequence value. We call this risk structure **r** and define it more precisely as follows:

- the likelihood structure of **r** is **p** as defined in Example 1;
- the set of consequence values are defined by real values from 1 to 5;
- the set of risks R is defined as the set of all non-negative real numbers;
- the operator $\odot$ is defined by $r \odot r' \triangleq (r - r')^2$;
- the risk value function is defined by $rv(l, c) \triangleq l * c$.

In the risk graph shown in Fig. 1, we have annotated the nodes S_5 and S_6 with the consequence values 2 and 5, respectively.

3 Test Procedure Identification

In this section, we discuss how a risk graph can be used as the basis for identifying test procedures. In particular, we are interested in finding a systematic technique for translating the risk graph into a list of (potential and as of yet not prioritized) textual descriptions that can be used as a starting point for (manual) refinement into a detailed test procedure that will eventually result in execution of test cases. Hence, our notion of a test procedure is more precisely an initial or a high-level test procedure.

A risk graph can be seen as a set of statements about the world. We take the position that testing a risk graph corresponds to checking the degree to which its statements are true. Consequently, every statement X of a risk graph corresponds to a test procedure description of the form *Check the degree to which X is true*, or just *Check X* for brevity.

We discuss three kinds of statements that are expressed by a risk graph.

Node statements A node in a risk graph is a statement about the likelihood of occurrence of an event. The meaning of each node p with likelihood l can be expressed by a statement of the form *p occurs with likelihood l*. In many risk assessment languages, nodes often correspond to threat scenarios, misuse cases, or unwanted incidents and they are labeled with a textual description for explanatory purposes. For instance, a node with likelihood l might be given the description *SQL injection successful* (as in Fig.2), in which case its meaning can be expressed by the statement *SQL injection successful occurs with likelihood l*.

Edge statements An edge in a risk graph is a statement about the likelihood of its target node occurring given that its source node has occurred. The meaning of an edge (p, q) with likelihood l is therefore a statement of the form: *p leads to q with conditional likelihood l*. In risk assessment languages, it is often the case that some kind of vulnerability has to be exploited, or some fault has to occur in order to get from p to q. Therefore checking statements about edges would often also involve checking for vulnerabilities or faults.

Path statements A path is a sequence of edges, starting with an initial node and ending up in a final node. The statement derived from a path is a concatenation of the statements derived from its edges, while injecting the work *then* between each edge statement.

All three statement kinds are possible starting points for a test procedure identification. However, our experience suggests that taking the edges as the starting point is the best choice. The problem with using the nodes as a starting point is that the likelihood of a node p in a risk graph depends on all edges and nodes that may lead up to p; to test the degree to which the degree to which the likelihood of p is true would in many cases amount to checking the degree to which the likelihoods of all its proceeding nodes and edges are true. This could result in a lack of traceability between the risk graph and the things that are actually tested.

The reason why we prefer edges over paths is that a path is just a sequences of edges, so by checking each edge in the path separately, we are able to check the entire path anyway. Although a path can provide a more precise characterization of the likelihood that the source p of an edge (p, q) can occur, this characterization has not been needed in practice in our experience.

For a given risk assessment language which may be seen as an instance of a risk graph, we may derive more descriptive statements by taking into account particularities of the language. We show an example of this in Sect. 6.

Example 5. In our technique, the two test procedures that can be derived from the edges $S_1 \xrightarrow{2/3} S_2$ and $S_1 \xrightarrow{1/4} S_3$ shown in Fig. 1 are: *Check that S_1 leads to S_2 with likelihood $\frac{2}{3}$*, and *Check that S_1 leads to S_3 with likelihood $\frac{1}{4}$*.

4 Test Procedure Prioritization

In this section, we describe a function for prioritizing test procedures on the basis of risk graphs. We take the position that every edge in a risk graph corresponds to a potential test procedure. Since there is a one to one correspondence between edges and test procedures, we will sometimes use the terms interchangeably. Hence, when we talk about prioritization of edges, we are also talking about prioritization of test procedures.

In order to prioritize a test procedure of the form *Check* X, where X is a statement about the world derived from a risk graph G, we take two notions into account:

Severity This is an estimate of the impact of whether X is true or not has on the risks of G. The intuition is that a high degree of impact should result in a high priority. In the extreme case, whether statement X is true or not has zero impact on the risk graph, therefore there no point in checking the degree to which X is true.

Confidence This is an estimate of how confident we are about the correctness of X. Intuitively, the less confident we are about the correctness of X, the more it makes sense to test it. In the extreme case, if we are completely confident in the correctness of X, then there is no point in checking whether X is true, because then we strongly believe that this will not give us any new information.

In some cases, the likelihood structure can be used for expressing confidence. For instance, if we use a likelihood structure whose likelihood values are intervals then we can use the size of the intervals as a measure of confidence. However, in general, the likelihood values cannot always be used to express confidence. For this reason, we introduce the notion of a confidence annotation that we can use to express the notion of confidence precisely and to annotate risk graphs with confidence values.

Definition 7. *(Confidence annotation) A confidence annotation* ϕ *is a tuple* (E, λ, Z, ci, z) *consisting of*

- *a set of edges* E *and a likelihood structure* λ*;*
- *a set of confidence values* Z*;*
- *a confidence interval function* $ci \in (\lambda.L \times Z) \rightarrow (\lambda.L \times \lambda.L)$ *mapping a likelihood value and a confidence value into a pair of likelihood values* (l, l') *referred to as a confidence interval, where* l *is the minimum confidence value and* l' *is the maximum confidence value.*
- *a function* $z \in E \rightarrow Z$ *mapping edges to confidence values.*

Given a risk graph G over ρ and confidence annotation ϕ whose edges are the same as $G.E$, and a test procedure derived from an edge e in G, we can now calculate the priority of the test procedure as follows:

– Calculate the confidence interval $ci(l(e), z(e)) = (l_{min}, l_{max})$ of edge e;
– Construct a risk graph G_{min} (the best case risk graph) obtained by replacing l with a minimum likelihood value l_{min}, and then recalculating all the likelihood values of the nodes using Def. 3.
– Construct a risk graph G_{max} (the worst case graph) in the same way by replacing l with a maximum likelihood value l_{max}.
– Add up the difference between the risk values of each node of G_{min} and G_{max}.

In the following, we define this precisely.

Definition 8. *(Priority) Given a risk graph G over risk structure ρ and confidence annotation ϕ with the same edges as G, the priority of an edge e, denoted $p[G, e]$, is defined as follows*

$$p[G, e] \triangleq \sum_{p \in G.Q_c} rv(l[G_{max}, p], c(p)) \odot rv(l[G_{min}, p], c(p))$$

where $G_{min} = we(G, e, l_{min})$ and $G_{max} = we(G, e, l_{max})$ for $(l_{min}, l_{max}) = ci(l(e), z(e))$.

The function is lifted to a set of edges E such that $p[G, E]$ yields the sum of the priorities in E, i.e.

$$p[G, E] \triangleq \sum_{e \in E} p[G, e]$$

Example 6. In Fig. 1, we have illustrated a risk graph annotated with confidence values. The confidence values are taken from a confidence annotation whose:

– edges E is the edges of the risk graph represented in Fig.1;
– likelihood structure is $\mathbf{p}$ as defined in Ex. 1;
– confidence values Z is defined by $\{Low, Medium, High\}$;
– confidence interval function ci is defined by $ci(l, Low) \triangleq int(l, 0.05)$, $ci(l, Medium) \triangleq int(l, 0.125)$, and $ci(l, High) \triangleq int(l, 0.25)$, where int is a function defined by $int(l, n) \triangleq (max(l - n, 0), min(l + n, 1)$;
– confidence annotation function z is defined as in Fig. 1.

Assume that we want to calculate the priority of the edge $S_1 \xrightarrow{2/3, Low} S_2$ in Fig. 1. Following the steps of the procedure, we:

– Calculate the confidence interval $ci(\frac{2}{3}, Low) = (0.617, 0.717)$.
– Replace the likelihood of the edge by its minimum likelihood 0.617, and recalculate the likelihood values of the risk graph. For the risk nodes S_5 and S_6, we get likelihood values 0.532, and 0.176, respectively;
– Calculate the maximum risk graph by replacing the edge likelihood by the maximum likelihood 0.717. For the risk nodes S_5 and S_6, we get likelihood values 0.599 and 0.176, respectively.

- Calculate the risk node difference. For S_5 we get $rv(0.532, 2) \odot rv(0.599, 2) = ((0.532{*}2)-(0.599{*}2))^2 = 0.018$. For S_6 we get $rv(0.176, 5) \odot rv(0.176, 5) = 0$. This yields a priority of 0.018 for edge $S_1 \rightarrow S_2$.

Similarly, we can calculate the priority of edge $S_1 \rightarrow S_3$. This yields a priority of 0.204. Hence edge $S_1 \rightarrow S_3$ has a higher priority than edge $S_1 \rightarrow S_2$.

5 Test Procedure Selection

In the previous section, we defined a function for prioritizing each edge in a risk graph under the assumption that each edge represented a potential test procedure. In this section, we define a technique for selecting the test procedures that should be further refined into detailed test procedures.

A simple technique would be to select all test procedures that have a higher priority than some priority threshold. However, it is often not the case that every potential test procedure represented by a risk graph can be refined into meaningful test cases given the scope and focus of the testing, and the knowledge, effort and tools available by the testing team. Furthermore, the time and resources required to implement a test procedure can vary significantly. To take this into account we need more information than what is expressed in the risk graph. We therefore introduce a notion of effort annotation.

Definition 9. *(Effort annotation) An effort annotation ea is a triple* (E, ef, max) *consisting of*

- *a set of edges E;*
- *a function $ef \in E \rightarrow \mathbb{R}$ mapping edges to effort estimates expressed as real values;*
- *a number $max \in \mathbb{R}$ representing the maximum effort available for testing.*

The effort annotation allows us assign effort estimates to edges in a graph, indicating the effort that it will take to implement and perform the corresponding test procedure. If an edge in a graph is not assigned any effort by the effort annotation, then we assume that the edge is not considered for test selection. E.g., if a given test procedure is completely out of scope then we can indicate this by not assigning any effort estimates to it.

A test procedure selection derived from a risk graph G is a subset of the edges of G. We say that a test selection E of G is valid w.r.t. to effort annotation ea, if all edges in E are assigned to an effort estimate by ea and the sum of effort estimates of E is less than or equal to the maximum effort of ea. This is precisely defined in the following.

Definition 10. *(Valid test procedure selection) Given a graph G over an effort assignment ea, we say that a set of edges E is a valid test procedure selection for a risk graph G under ea, denoted $vs[G, E]$, if*

$$E \subseteq G.E \cap ea.E \qquad \wedge \qquad (\sum_{e \in E} ea.ef(e)) \leq ea.max$$

A test procedure selection for a risk graph G is considered optimal if it is a valid selection of G and there are no other valid selections with a higher priority. This is formally defined in the following.

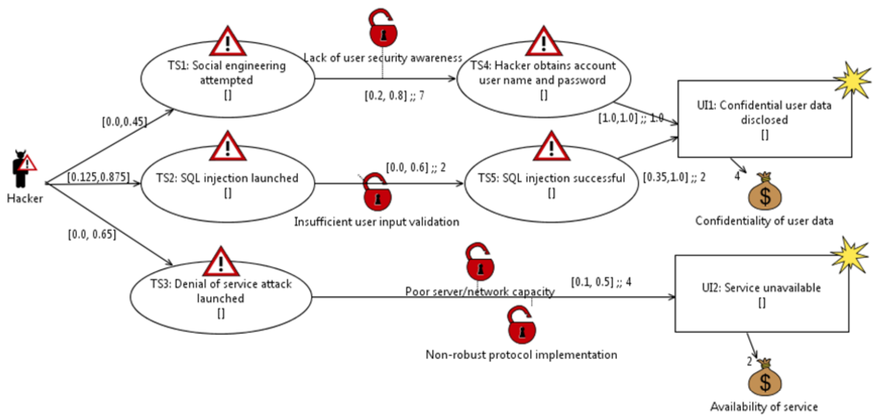
Definition 11. *(Optimal test procedure selection) Given a risk graph G over an effort estimate ea, we say that the set of edges E is an optimal test procedure selection for G, denoted $os[G, E]$, if*

$$vs[G, E] \wedge (\forall E' \mid vs[G, E'] \Rightarrow p[G, E'] \leq p[G, E])$$

Example 7. Continuing Ex. 6, assume an effort assignment ea whose edges $ea.E$ is the set of all edges going from S_1; whose effort function assigns the number 2 to each edge in $ea.E$; whose maximum available effort $ea.max$ is 4. Since the sum of the effort assignment of the three edges in $ea.E$ is greater than $ea.max$, $ea.E$ is not a valid test selection. In this case, a valid test procedure selection can only contain two edges, and the optimal test procedure selection is in this case the two edges going from S_1 with the highest priority.

6 An Extended Example

Up to this point, we have only exemplified our technique on risk graphs. In this section, we demonstrate our technique on the CORAS threat diagram language. An example of a CORAS threat diagram is shown in Fig. 2. The diagram depicts some scenarios in which a hacker can disclose confidential user data or cause a service to become unavailable.



TS1: Social engineering attempted []
Lack of user security awareness
[0.2, 0.8] ;; 7
TS4: Hacker obtains account user name and password []
[0.0,0.45]
[1.0,1.0] ;; 1.0
UI1: Confidential user data disclosed []
Hacker
[0.125,0.875]
TS2: SQL injection launched []
[0.0, 0.6] ;; 2
TS5: SQL injection successful []
[0.35,1.0] ;; 2
4
Insufficient user input validation
Confidentiality of user data
[0.0, 0.65]
TS3: Denial of service attack launched []
Poor server/network capacity
[0.1, 0.5] ;; 4
UI2: Service unavailable []
Non-robust protocol implementation
2
Availability of service

Fig. 2. Example of a CORAS threat diagram

A CORAS diagram can be seen as an instance of a risk graph, but there are some minor differences. In CORAS, a node may be of one of three kinds: A *threat*, a *threat scenario*, or an *unwanted incident*. Examples of these are in Fig. 2 the nodes labeled 'Hacker', 'Social engineering attempted', and 'Confidential user data disclosed', respectively. There are two kinds of annotations in CORAS: *vulnerabilities* and *assets*. Vulnerabilities are shown as open locks and may be attached to edges, while assets are shown as money bags and are attached to unwanted incidents. A label on the edge between an asset and an unwanted incident denotes the consequence value of the unwanted incident, e.g. the unwanted incident 'Confidential user data disclosed' in Fig. 2 has a consequence value of 4.

As with risk graphs, nodes and edges may be assigned likelihoods. In CORAS diagrams, the likelihoods (if any) are often written inside square brackets. The empty bracket [] means that a node does not have a likelihood. In addition to this, we have also annotated effort estimates on some of the edges (although this is not standard CORAS convention), i.e. an edge whose label has the form $l;;n$ means that the edge has a likelihood of l and an effort estimate of n.

In the diagram shown in Fig. 2, likelihood values are given as probability intervals, and we make no assumption about the independence of the occurrence of nodes in the diagram. In particular, we assume a likelihood structure **d** whose

- set of likelihood values is the set of all pairs (n, n') of real numbers between 0 and 1 such that n is less than or equal to n', i.e. $L \triangleq \{(n, n') \in \{0, \dots, 1\} \times \{0, \dots, 1\} \,|\, n \leq n'\}$;
- maximum value is $(1, 1)$ and the minimum value is $(0, 0)$;
- or-operator is defined as follows: $(n, n') \oplus (m, m') \triangleq (max((n, n'), (m, m')), (n+m, n'+m'))$, where max is function that takes two intervals and returns the maximum of those, i.e., $max((n, n'), (m, m'))$ yields (n, n') if $n' > m'$ or $n' = m' \wedge n \geq m$, otherwise it yields (m, m');
- and-operator is defined by interval multiplication i.e. $(n, n') \otimes (m, m') \triangleq (n * m, n' * m')$.

In addition, we assume the risk structure **rd**, whose

- likelihood structure is **d** as defined above;
- consequence values are defined by real values from 1 to 5;
- risk values R is the set pairs of all non-negative real numbers;
- difference operator $\odot$ is defined by $(n, n') \odot (m, m') \triangleq (n - m)^2 + (n' - m')^2$;
- risk value function is defined by $rv((n, n'), c) \triangleq (n * c, n' * c)$.

Since we are using likelihood intervals, we can use the size of the intervals as an estimate of confidence, thus there is no need to annotate the edges with additional confidence estimates. Formally, we use the confidence annotation **ca** whose:

- edges E is the set of edges of the risk graph represented in Fig. 2;
- likelihood structure is **d** defined above;
- confidence values Z is defined by $\{\perp\}$;
- confidence interval function ci is defined by $ci(l, \perp) = l$;
- confidence annotation function z annotates all edges by $\perp$.

CORAS threat diagrams contain more information than risk graphs, and this information can be used in order to define a more meaningful translation of edges to textual descriptions representing test procedures. In fact, CORAS threat diagrams already have a translation into English text [13]. We can use this in our technique for test procedure identification and prioritization.

The result of using the test procedure technique on the risk graph represented in Fig. 2 over the risk structure **rd** and with the confidence annotation **ca** is shown in Table. 1. As discussed previously, many of the edges of the risk graph represented in Fig. 2 have been annotated with estimate values. These values are to be understood as an estimate of the number of days it will take to refine and perform the test procedure corresponding to a given edge. For instance, performing the test procedure with the highest priority in Table. 1 is estimated to take two days. Not all edges have been annotated with efforts. Edges without effort annotation are understood as being out of scope. As shown in Fig. 2, the confidence regarding the likelihood of whether a hacker will launch an SQL attack in the first place is fairly low (ranging from 0.125 to 0.875). This contributes in giving the corresponding test procedure a high priority. However, these kinds of test procedures can be difficult to perform, at least using conventional security testing techniques, hence it is in this case excluded from the test selection. Given that an attack is performed however, it is often possible to test whether the system has any vulnerabilities that can be exploited by the attack. These kinds of test procedures have been given effort estimates.

Table 1. List of prioritized test procedures

Test procedure	Priority	Effort
Check that SQL injection launched leads to SQL injection successful with conditional likelihood [0.0, 0.6], due to vulnerability Insufficient user input validation.	**4.421**	**2 days**
Check that Hacker initiates SQL injection launched with likelihood [0.125, 0.875].	3.240	N / A
Check that Hacker initiates Social engineering attempted with likelihood [0.0, 0.45].	2.203	N / A
Check that SQL injection successful leads to Confidential user data disclosed with conditional likelihood [0.35, 1.0].	**1.863**	**2 days**
Check that Social engineering attempted leads to Hacker obtains account user name and password with conditional likelihood [0.2, 0.8], due to vulnerability Lack of user security awareness.	1.166	7 days
Check that Hacker initiates Denial of service attack launched with likelihood [0.0, 0.65].	0.439	N / A
Check Denial of service attack launched leads to Service unavailable with conditional likelihood [0.1, 0.5], due to vulnerabilities Poor server/network capacity and Non-robust protocol implementation.	0.270	4 days
Check that Hacker obtains account user name and password leads to Confidential user data disclosed with conditional likelihood [1.0, 1.0].	0.000	1 day

If we assume that we have a maximum of seven days available for testing, then an optimal test procedure selection are the two test procedures shown in bold face in Table. 1.

7 Related Work

Although there are several approaches that use risk assessment to identify and prioritize tests, most of these approaches are based on already existing techniques from risk assessment and testing. Very few new techniques are proposed that specifically combine risk assessment and testing.

Almost all the approaches to risk-based testing use risk assessment in one of two ways. Either (I) the risk assessment is used to prioritize those parts/features of the system under test that are most risky, or (II) risk assessment is used to identify tests (often as part of a failure/threat identification process).

The approaches that we know of that fall into category (I) are Bach [1], Redmill [17,15,16], Souza et al. [21,20], Bai et al. [2], Felderer et al [7], Rosenberg et al. [18], and Wong et al. [23]. All of these approaches use already existing techniques such as HAZOP [10] to identify risks, or code complexity measures to identify areas of code that are most likely to fail. In addition, they do not use risk assessment for the purpose of risk identification.

The approaches that we are aware of that fall into category (II) are Murthy et. al. [14], Zech et al. [25,24], Casado et al. [4], Kumar et al. [12], and Gleirscher [8]. None of these approaches use the risk assessment results for test prioritization.

Of the two categories of risk-based approaches, our technique fits best into category (II). However, it is unique in that it takes test prioritization into account. Furthermore, unlike the techniques used by the approaches that are cited above, it has been particularly developed for the purpose of risk-based testing.

The only approaches that we are aware of that present novel techniques that are specifically intended to be used in a risk-based testing setting are Chen et. al [6], Kloos et. al. [11], and Stallbaum et. al. [22]. All of approaches assume that a test model is already available at the start of the process. They then discuss how to update/annotate the test model based on risk information, and how this can be used to prioritize test cases. This process is very different from the process our technique is intended to support, since we do not assume that a test model is available.

8 Conclusion and Future Work

We have presented a technique for risk-based test procedure identification, prioritization, and selection. The technique takes a risk graph (with some additional annotations) as input, and yields a prioritized list of initial test procedures as output. The technique is general in the sense that it can be used with many kinds of risk documentation languages, and many kinds of likelihood and risk types.

A risk graph is as set of nodes and edges that can be seen as a set of statements about the likelihood of occurrence of events, how the events are related, and the consequence of events occurring. We have argued that testing an element of the risk graph corresponds to checking the degree to which its statement about the world is correct. Our notion of a test procedure is then a textual description of the form *Check the degree to which X is true* where X is a statement about the world derived from the risk graph. These kinds of test procedures are meant as starting points for refinement into more detailed test procedures that will eventually be used to derive concrete test cases.

We have defined a technique for test procedure prioritization that is based on two notions: severity and confidence. Severity is an estimate of the impact of the statement being tested has on the risk values, and confidence is an estimate of how confident we are about the correctness of the statement. In the paper, we have defined these notions precisely, and based on this, defined a test procedure prioritization function that can be automated. Finally, we have defined a technique for test procedure selection which is based on a notion of effort.

Although risk-based testing has been discussed a lot in the literature, most approaches rely on already existing techniques from risk assessment which are not specific to risk-based testing. Furthermore, most approaches to risk-based testing either address risk-based test identification or risk-based test prioritization. Our technique is unique in that it addresses both these issues.

The technique described in the paper only works if we assume that all likelihood values of edges are known. However, our experience suggests that this is not always the case. In fact, we more often have the likelihoods for the nodes. In future work, we will address this issue. In addition, as part of future work, we will define a visualization technique for showing the priority of a test procedure in terms of its impact on the risk picture. Currently, priorities are only given as real values which only give a sense of the relative importance of the test procedures, but not much else. We have already implemented the technique for test prioritization. However, we plan to integrate this into the CORAS tool for risk assessment, and this is still ongoing work.

Acknowledgments. This work has been conducted as a part of the EU project RASEN (316853) funded by the European Commission within the 7th Framework Program.

References

1. Bach, J.: Heuristic risk-based testing. Software Testing and Quality Engineering Magazine 11, 9 (1999)
2. Bai, X., Kenett, R.S.: Risk-based adaptive group testing of semantic web services. In: Proc. of the 33rd Annual IEEE International Computer Software and Applications Conference (COMPSAC), pp. 485–490. IEEE Computer Society (2009)
3. Brændeland, G., Refsdal, A., Stølen, K.: Modular analysis and modelling of risk scenarios with dependencies. Journal of Systems and Software 83(10), 1995–2013 (2010)
4. Casado, R., Tuya, J., Younas, M.: Testing long-lived web services transactions using a risk-based approach. In: Proc. 10th International Conference on Quality Software (QSIC), pp. 337–340. IEEE Computer Society (2010)
5. Charniac, E.: Bayesian networks without tears: making bayesian networks more accessible to the probabilistically unsophisticated. AI Magazine 12(4), 50–63 (1991)
6. Chen, Y., Probert, R.L., Sims, D.P.: Specification-based regression test selection with risk analysis. In: Proc. of the 2002 Conference of the Centre for Advanced Studies on Collaborative Research, CASCON 2002, p. 1. IBM Press (2002)

7. Felderer, M., Haisjackl, C., Breu, R., Motz, J.: Integrating manual and automatic risk assessment for risk-based testing. In: Biffl, S., Winkler, D., Bergsmann, J. (eds.) SWQD 2012. LNBIP, vol. 94, pp. 159–180. Springer, Heidelberg (2012)
8. Gleirscher, M.: Hazard-based selection of test cases. In: Proc. of the 6th International Workshop on Automation of Software Test, pp. 64–70. ACM (2011)
9. International Electrotechnical Commission. Event Tree Analysis in Dependability Management - Part 3: Application Guide - Section 9: Risk Analysis of Technological Systems. IEC 60300 (1990)
10. International Electrotechnical Commission. IEC 61025 Fault Tree Analysis, FTA (1990)
11. Kloos, J., Hussain, T., Eschbach, R.: Risk-based testing of safety-critical embedded systems driven by fault tree analysis. In: Proc. of IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops (ICSTW), pp. 26–33. IEEE (2011)
12. Kumar, N., Sosale, D., Konuganti, S.N., Rathi, A.: Enabling the adoption of aspects - testing aspects: A risk model, fault model and patterns. In: Proc. of the 8th ACM International Conference on Aspect-oriented Software Development, AOSD 2009, pp. 197–206. ACM (2009)
13. Lund, M.S., Solhaug, B., Stølen, K.: Model Driven Risk Analysis - The CORAS Approach. Springer (2011)
14. Murthy, K.K., Thakkar, K.R., Laxminarayan, S.: Leveraging risk based testing in enterprise systems security validation. In: Proc. of the First International Conference on Emerging Network Intelligence, pp. 111–116. IEEE Computer Society (2009)
15. Redmill, F.: Exploring risk-based testing and its implications: Research articles. Softw. Test. Verif. Reliab. 14(1), 3–15 (2004)
16. Redmill, F.: Theory and practice of risk-based testing. Software Testing, Verification and Reliability 15(1), 3–20 (2005)
17. Redmill, F., Chudleigh, M.F., Catmur, J.R.: Principles underlying a guideline for applying HAZOP to programmable electronic systems. Reliability Engineering and System Safety 55(3), 283–293 (1997)
18. Rosenberg, L., Stapko, R., Gallo, A.: Risk-based object oriented testing. In: Proc. of the 24th Annual Software Engineering Workshop. NASA (1999)
19. Schneider, B.: Attack trees: modeling security threats. Dr. Dobb's Journal of Software Tools 24(12), 21–29 (1999)
20. Souza, E., Gusmão, C., Venancio, J.: Risk-based testing: A case study. In: Proc. of ITNG, pp. 1032–1037. IEEE Computer Society (2010)
21. Souza, E., Gusmão, C., Venancio, J., Alves, K., Melo, R.: Measurement and control for risk-based test cases and activities. In: Proc. of Test Workshop (LATW 2009), pp. 1–6. IEEE (2009)
22. Stallbaum, H., Metzger, A., Pohl, K.: An automated technique for risk-based test case generation and prioritization. In: Proc. of the 3rd International Workshop on Automation of Software Test, pp. 67–70. ACM (2008)
23. Wong, W.E., Qi, Y., Cooper, K.: Source code-based software risk assessing. In: Proc. of the 2005 ACM Symposium on Applied Computing, SAC 2005, pp. 1485–1490. ACM (2005)
24. Zech, P., Felderer, M., Breu, R.: Towards a model based security testing approach of cloud computing environments. In: 2012 IEEE Sixth International Conference on Software Security and Reliability Companion (SERE-C), pp. 47–56. IEEE (2012)
25. Zech, P., Felderer, M., Breu, R.: Towards risk - driven security testing of service centric systems. In: QSIC, pp. 140–143. IEEE (2012)

A Risk Assessment Framework for Software Testing

Michael Felderer, Christian Haisjackl, Viktor Pekar, and Ruth Breu

Institute of Computer Science, University of Innsbruck, Austria
{michael.felderer,christian.haisjackl,viktor.pekar,ruth.breu}@uibk.ac.at

Abstract. In industry, testing has to be performed under severe pressure due to limited resources. Risk-based testing which uses risks to guide the test process is applied to allocate resources and to reduce product risks. Risk assessment, i.e., risk identification, analysis and evaluation, determines the significance of the risk values assigned to tests and therefore the quality of the overall risk-based test process. In this paper we provide a risk assessment model and its integration into an established test process. This framework is derived on the basis of best practices extracted from published risk-based testing approaches and applied to an industrial test process.

Keywords: Risk Assessment, Risk Identification, Risk Analysis, Risk Evaluation, Risk-Based Testing, Risk Management, Software Testing.

1 Introduction

Risk-based testing (RBT) is a pragmatic and well-known approach to address the problem of ever limited testing resources that recently gained much attention [1]. It is based on the intuitive idea to focus test activities on those scenarios that trigger the most critical situations for a software system [2]. Its appropriate application may then have several benefits. RBT optimizes the allocation of resources (budget, time, persons), is a means for mitigating risks, helps to early identify critical areas, and provides decision support for the management. Risk-based testing involves the identification, analysis and evaluation of product risks, which are together referred to as risk assessment, and the use of risks to guide the test process [3].

Because risk identification, analysis, and evaluation determine the significance of the risk values assigned to tests and therefore the quality of the overall test process, they are core activities in every risk-based test process. Although several RBT approaches are available [4], and the upcoming international standard ISO/IEC 29119 [5] on testing techniques, processes, and documentation even requires the consideration of risks as an integral part of the test planning process, a framework on how to integrate risk assessment in a test process has not been proposed. But such a framework provides guidelines and supports test and process managers to establish a risk-based test process on the basis of an existing test process.

The objective of this paper is to provide a framework for integrating risk assessment, i.e., risk identification, analysis, and evaluation, into an established test process. The framework contains a risk assessment model which configures the risk-based test process. It is derived on the basis of best practices extracted from published RBT approaches and applied to an industrial test process.

The remainder of this paper is structured as follows. Section 2 discusses background on risk-based testing and related work. Section 3 defines a risk assessment framework for testing purposes. Section 4 shows how this model is applied in industrial projects. Finally, Section 5 concludes the paper and presents future work.

2 Background on Risk-Based Testing

Risk-based Testing (RBT) is a type of software testing that considers risks assigned to risk items for testing activities [6,7]. In risk-based testing, testing activities are supported by risk management activities. It therefore integrates a risk management process into a test process. In this section we discuss background on the concept of risk (Section 2.1), test and risk management processes (Section 2.2) as well as RBT approaches (Section 2.3).

2.1 Concept of Risk

A risk is the chance of injury, damage or loss and typically determined by the probability of its occurrence and its impact [8]. As it is the chance of something happening that will have an impact on objectives [9], the standard risk formalization [3] is based on the two factors probability (P), determining the likelihood that a failure assigned to a risk occurs, and impact (I), determining the cost or severity of a failure if it occurs in operation. Mathematically, the risk exposure R of an arbitrary asset a, i.e., something to which a party assigns value, is determined based on the probability P and the impact I in the following way:

$$R(a) = P(a) \circ I(a)$$

In the context of testing, assets are arbitrary testable artifacts also called risk items. For instance, requirements, components, security risks or failures are typical risk items to which risk exposure values R as well as tests are assigned. Within testing, a risk item is assigned to test cases which are typically associated with risk exposure values themselves derived from the risk items' risk exposure values. Risk exposure is sometimes also called risk coefficient, risk value or not distinguished from the risk itself. The depicted operation $\circ$ represents a multiplication of two numbers or a cross product of two numbers or letters (and can principally be an arbitrary mathematical operation used to determine risk). The factors P and I may be determined directly via suitable metrics or indirectly via intermediate criteria based on the Factor-Criteria-Metric model [10]. The probability typically considers technical criteria like complexity of components assigned to the risk item and the impact considers business criteria like monetary loss. The metrics can be measured automatically, semi-automatically or manually. For instance, the complexity of a component can be estimated automatically by the McCabe complexity and the monetary loss can be estimated manually by a customer. Based on the determined metrics, risk exposure values are computed on the basis of a calculation procedure. Finally, risk exposure values are assigned to risk levels. A risk level [3] indicates the criticality of risk items and serves the purpose to compare risk items as well as to determine the use of resources, e.g., for testing. Risk levels are often defined via risk matrices combining probability and impact of a risk. An example for a risk matrix is shown in Fig. 1.

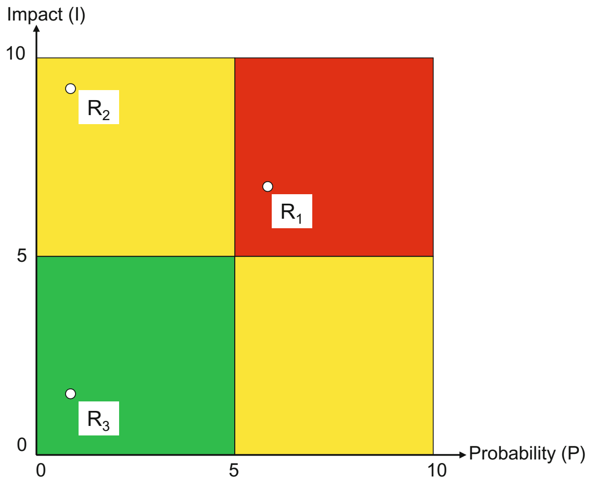


Fig. 1. Risk Matrix Example

The 2x2 risk matrix of Fig. 1. Probability and impact range from 0 to 10 and are shown on the x-axis and y-axis, respectively. Items in the lower left cell ([0..5] × [0..5]) have low risk, items in the upper right cell ([5..10] × [5..10]) have high risk, and items in the remaining cells ([0..5] × [5..10] and [5..10] × [0..5]) have medium risk. For instance, risk R_1 in Figure 1 with value 6×7 is high, R_2 with value 1×9 is medium, and R_3 with value 1×2 is low.

2.2 Basic Concepts of Test and Risk Management Processes

A *test process* contains the core activities *test planning*, *test design*, *test implementation*, *test execution* as well as *test evaluation* [3]. Test planning is the activity of establishing or updating a test plan. A test plan is a document describing the scope, approach, resources, and schedule of intended test activities [3]. During the test design phase the general testing objectives defined in the test plan are transformed into tangible test conditions and test cases. Tests are then implemented which contains remaining tasks like preparing test harnesses and test data, or writing automated test scripts which are necessary to enable the execution of the implementation-level test cases. The tests are then executed and all relevant details of the execution are recorded in a test log. During the test evaluation and reporting phase, the exit criteria are evaluated and the logged test results are summarized in a test report. Development projects typically contain several test cycles and therefore all or some phases of the test process are performed iteratively.

A *risk management process* contains the core activities *risk identification*, *risk analysis*, *risk evaluation*, *risk treatment*, and *risk monitoring* [9]. In the risk identification phase risk items are identified. In the risk analysis phase the probability and impact of risk items and hence their risk exposure values are estimated. In the risk evaluation phase, the significance of risk is assessed based on the estimated risk exposure values. As a consequence, risk items may be assigned to risk levels defining a risk classification and a prioritization. In the risk treatment phase actions for obtaining a satisfactory situation are determined and implemented. In case of risk-based testing, testing is applied as a measure to treat risks. In the risk monitoring phase risks are tracked over time and their status is reported. In addition, the effect of the implemented actions is determined. The activities risk identification, risk analysis, and risk evaluation are often collectively referred to as *risk assessment*, while the activities risk treatment and risk monitoring are referred to as *risk control.* As in the context of RBT, testing is per definition applied for risk control, only risk assessment, i.e., risk identification, analysis, and evaluation, has to be integrated into the test process as a separate activity.

2.3 Risk-Based Testing Approaches

The overall purpose of RBT approaches is to test in an efficient and effective way driven by risks. As mentioned before, every available risk-based testing approach therefore integrates testing and risk assessment activities. Several RBT approaches have been proposed in scientific conferences and journals. We systematically extracted these approaches from comprehensive related work sections of four recently published journal articles on risk-based testing [11,4,12,13] to get a broad and representative overview of RBT approaches. We considered all RBT approaches defined in the journal articles themselves as well as all RBT approaches cited in at least one related work section of the four journal articles. To guarantee evidence of the approaches and enough details to extract relevant information, we considered only RBT approaches reported in papers with a length of at least four pages published in a scientific journal or in conference proceedings. Table 1 lists all collected RBT approaches ordered by the date of their first publication. Some approaches, i.e., Redmill, Stallbaum, Souza, as well as Felderer and Ramler are covered by more than one cited publication (see entries with identifiers 03, 04, 05 and 13 in Table 1). Most listed approaches are cited by more than one journal article which is an additional indicator for the relevance of the RBT approaches collected in Table 1.

Table 1. Overview of Identified Risk-based Testing Approaches

ID	Approach	Description
01	Amland [6]	The approach defines a process which consists of the steps (1) planning, (2) identification of risk indicators, (3) identification of cost of a fault, (4) identification of critical elements, (5) test execution as well as (6) estimation to complete. In addition, it is presented how the approach was carried out in a large project.
02	Chen et al. [14]	The approach defines a specification-based regression test selection with risk analysis. Each test case is a path through an activity diagram (its elements represent requirements attributes) and has an assigned cost and severity probability. The test selection consists of the steps (1) assessment of the cost, (2) derivation of severity probability, and (3) calculation of risk exposure for each test case as well as (4) selection of safety tests. The risk exposure of test cases grouped to scenarios is summed up until one runs out of time and resources. The approach is evaluated by comparing it to manual regression testing.
03	Redmill [15,16]	The approach reflects on the role of risk for testing in general and proposes two types of risk analysis, i.e., single-factor analysis based on impact or probability as well as two-factor analysis based on both factors.
04	Stallbaum et al. [17,18]	The approach is model-based. Risk is measured on the basis of the Factor-Criteria-Metrics model and annotated to UML use case and activity diagrams from which test cases are derived.
05	Souza et al. [19,20]	The approach defines a risk-based test process including the activities (1) risk identification, (2) risk analysis, (3) test planning, (4) test design, (5) test execution, as well as (6) test evaluation and risk control. In addition, metrics to measure and control RBT activities are given. The approach is evaluated in a case study.
06	Zimmermann et al. [21]	The approach is model-based and statistical using Markov chains to describe stimulation and usage profile. Test cases are then generated automatically taking the criticality of transitions into account. The approach focuses on safety-critical systems and its application is illustrated by examples.
07	Kloos et al. [22]	The approach is model-based. It uses Fault Tree Analysis during the construction of test models represented as state machine, such that test cases can be derived, selected and prioritized according to the severity of the identified risks and the basic events that cause it. The focus of the approach are safety-critical systems and its application is illustrated by an example.
08	Yoon and Choi [23]	The approach defines a test case prioritization strategy for sequencing test cases. Each test case is prioritized on the basis of the product of risk exposure value manually determined by domain experts and the correlation between test cases and risks determined by mutation analysis. The effectiveness is shown by comparing the number and severity of faults detected to the approach of Chen et al.
09	Zech [24]	The approach is model-based and derives a risk model from a system model and a vulnerability knowledge base. On this basis a misuse case model is derived and test code generated from this model is executed. The approach is intended to be applied for testing cloud systems.
10	Bai et al. [11]	The approach addresses risk-based testing of service-based systems taking the service semantics which is expressed by an OWL ontology into account. For estimating probability and impact dependencies in the ontology are considered. The approach considers the continuous adjustment of software and test case measurement as well as of rules for test case selection, prioritization and service evaluation. The approach is evaluated by comparing its cost and efficiency to random testing.

Table 1. *(continued)*

11	Felderer et al. [7]	The approach defines a generic risk-based test process containing the steps (1) risk identification, (2) test planning, (3) risk analysis, (4) test design as well as (5) evaluation. Steps (2) and (3) can be executed in parallel. For this test process a risk assessment model based on the Factor-Criteria-Metrics model is defined. The metrics in this model can be determined automatically, semi-automatically or manually. The approach is illustrated by an example.
12	Wendland et al. [2]	The approach is model-based. It formalizes requirements as integrated behavior trees and augments the integrated behavior tree with risk information. Then for each risk an appropriate test directive is identified, and finally both the risk-augmented integrated behavior tree and the test directive definition are passed into a test generator.
13	Felderer and Ramler [12,25]	The approach defines a process to stepwise introducing risk-based testing into an established test process. On this basis four stages of risk-based test integration are defined, i.e., (1) initial risk-based testing including design and execution of test cases on the basis of a risk assessment, (2) risk-based test results evaluation, (3) risk-based test planning, as well as (4) optimization of risk-based testing. The approach is evaluated in a case study.
14	Ray and Mohapatra [13]	The approach defines a risk analysis procedure to guide testing. It is based on sequence diagrams and state machines. First one estimates the risk for various states of a component within a scenario and then, the risk for the whole scenario is estimated. The key data needed for risk assessment are complexity and severity. For estimating complexity inter-component state-dependence graphs are introduced. The severity for a component within a scenario is decided based on three hazard techniques: Functional Failure Analysis, Software Failure Mode and Effect Analysis and Software Fault Tree Analysis. The efficiency of the approach is evaluated compared to another risk analysis approach.

3 Risk Assessment Framework

In this section we present a risk assessment framework for risk-based testing purposes. This framework is shown in Fig. 3. It contains a *risk assessment model* which configures the *risk-based test process*. The execution of the test process provides feedback to continuously refine and improve the risk assessment model. As mentioned in the previous section, the risk-based test process integrates risk assessment into the test process and uses risks to support all phases of the test process, i.e., test planning, design, implementation, execution, and evaluation. The framework is based on the risk-based test process which is configured by and provides feedback for the risk assessment model and explained as background in Section 2.

The risk assessment model and its elements therefore determine the overall risk-based test process and are the main component of our risk assessment framework for testing purposes. The risk assessment model defines the test scope, the risk identification method, a risk model and the tooling for risk assessment. In the following, we explain these elements in more detail illustrated by examples from the RBT approaches collected in Section 2.3. Each mentioned approach is referred to by its name and identifier. For the often cited approach of Amland [6] we discuss all aspects of risk assessment model definition.

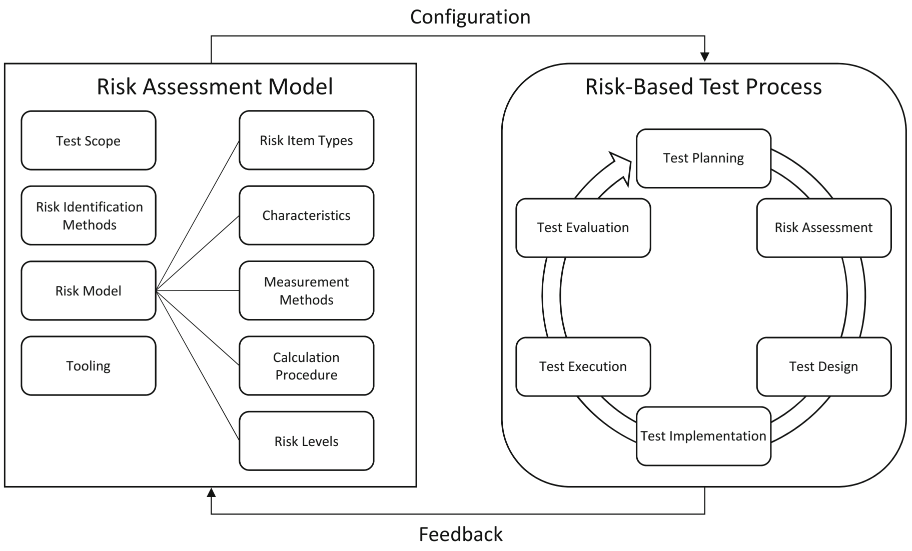


Fig. 2. Risk Assessment Framework

3.1 Test Scope

The test scope determines whether and how risk assessment is performed. It provides the overall testing context and typically considers the test object, test resources and test strategy. The test object defines the component or system to be tested and therefore influences the risk identification method and risk model. Limited test resources, i.e., personnel, time or budget, are typically the main driver for performing risk-based testing. Thus, the available resources determine whether a risk-based testing approach is required or not. The test strategy is a high-level description of the test levels to be performed and the testing within those levels determining risk-based testing as well.

All listed RBT approaches implicitly presume that prerequisites for the application of risk-based testing like limited resources are fulfilled and that the test objects are defined. Amland (01), for instance, states, “As for all projects, time and resources were limited.” (cf. [6], page 2). Furthermore, all approaches consider system or integration testing for components, services or complete systems.

3.2 Risk Identification Methods

Risk identification methods are techniques to identify risks items. There are several risk identification methods such as brainstorming, risk checklists, and failure history available [3,26] which can be applied and tailored to a specific RBT context to define a risk model. Different roles of the software engineering process like product managers, business analysts, software architects, testers or developers as well as different artifact like requirements specifications, documentation, defect databases or source code can be considered in specific risk identification methods.

Most RBT approaches do not explicitly mention the underlying risk identification methods but only present the resulting risk model and its application. Amland (01), for instance, explains the step 'identification of risk indicators' in which risk criteria are selected in a group meeting to guarantee that the used criteria are meaningful to those participating in the process of assessment. Souza et al. (05) use a taxonomy-based questionnaire answered by the project members, followed by a brainstorming meeting to identify technical risks.

3.3 Risk Model

The risk model is based on the concept of risk (see Section 2.1 the core artifact of the risk assessment model. It determines how the risk assessment is conducted in the risk-based test process. As Fig. 3 shows, the risk model consists of risk item types, characteristics, measurement methods, a calculation procedure, as well as risk levels which together define how risks are assessed. In the following, we explain these parts of the risk model in more detail.

Risk Item Types. The risk items type determines the risk items, i.e., the elements to which risk exposure values and tests are assigned, and their representation. For instance, Amland (01) assigns risks to system functions like 'Close Account' collected in a list. Furthermore, Chen et al. (02) assigns risks to test cases represented as path through an activity diagram, Zimmermann et al. (06) to critical functions represented as transitions in Markov chains, Kloos et al. (07) to safety risks represented as fault trees and state machines, Bai et al. (10) to web services represented as semantic models in OWL-S, and Wendland et al. (12) to requirements represented in behavior trees. Yoon and Choi (08) consider abstract sources of risk and assign the number of faults lying within the scope of a given risk and the test cases covering these faults to it. Felderer and Ramler (13) discuss different viewpoints for risk assessment, i.e., functional, architectural as well as development viewpoint, and conclude that the architectural viewpoint based on the components provides the most comprehensive structure in the considered project. Finally, Ray and Mohapatra (14) assigns risks to components represented as state machines, state dependence graphs and fault trees.

Characteristics. Characteristics define factors and their relationship to determine the risk. As such they define the applied risk concept. Typically, at least factors for probability and impact are considered which may be further refined based on the Factor-Criteria-Metrics model [27] defining a tree of factors with concrete measurable metrics at its leaves. Sometimes there are no defined characteristics and the risk is measured directly.

Amland (01) defines the factors probability and cost. For probability the criteria 'new functionality', 'design quality', 'size', and 'complexity' are distinguished, and for cost the criteria 'cost for customer' and 'fault occurrence'. Furthermore, Redmill (03) distinguishes between single-factor analysis based on impact or probability as well as two-factor analysis based on both factors. Stallbaum et al. (04) as well as Felderer et al. (11) define characteristics explicitly on the Factor-Criteria-Metrics model. Both distinguish the factors probability and impact and state that probability is mainly determined by technical criteria and metrics of software development activities but impact mainly by business criteria and metrics of domain analysis. Finally, Ray and Mohaptra (14) take the factors severity and complexity of components into account.

Measurement Methods. A measurement method defines how values are directly assigned to factors. The measurement can be performed manually or automatically. If the measurement is performed manually, the role performing the estimation and the procedure how the estimation is performed (e.g., a consensus meeting if several persons perform the estimation) have to be defined. If it is performed automatically, the measurement object and the measurement tool, e.g., a static analysis tool, have to be defined. Each measured factor requires a scale of arbitrary range for its assigned value. For manual measurement, a Likert scale is typically used where selection items are assigned to values. Automatically measured values are used directly or they are mapped to another scale.

Amland (01) applies a three-point Likert scale with low (1), medium (2) and high (3) or all criteria. The numeric values for the probability criteria were determined in a consensus meeting where the roles developer, designer, product specialist, quality manage, development manager, project manager, sales manager and corporate management were present. The cost criteria were determined by the customer and the supplier, respectively. Measurement methods similar to Amland (01) are applied in many industrial settings [12]. More advanced approaches are presented by Zech (09) who defines an automated approach to measure security risk values based on a system model and a vulnerability knowledge base.

Calculation Procedure. The calculation procedure defines how risk exposure values are calculated on the basis of other risk exposure values, characteristics, measured values and testing information. It determines how to aggregate values, i.e., which aggregation function to apply, how to scale values and how to weight different factors.

Amland (01) weights the values for the probability factors and computes the probability value as their weighted sum. The cost value is the average of the two cost factor values. The risk exposure value is then the product of the probability and cost value. Stallbaum et al. (04) determines the risk exposure value for each action (modeled in an activity diagram) by the product of the probability that an entity contains a fault and he total damage caused by this fault which are both measured on a five-level scale from 1 to 5. The risk exposure value of a test case, which is a sequence of actions, is then the sum of the actions' risk exposure values.

Risk Levels. Risk levels indicate the criticality of risk items and serve the purpose to compare risk items as well as to configure testing activities. Risk levels can be expressed either qualitatively or quantitatively [2]. For instance, numeric risk exposure values can be directly used as quantitative risk levels. Although, there is no restriction on the number of risk levels, a frequently used qualitatively scale for risk levels is low, medium, and high. Risk levels are often defined by the two dimensions probability and impact (each with levels low, medium and high) which are visualized in a risk matrix. Different areas in the risk matrix may then mapped to risk levels low, medium and high. If risk levels are measured qualitatively, factors are either directly measured qualitatively, e.g., with levels low, medium, and high, or their numeric values are mapped to these levels.

If risk levels are defined explicitly in the listed approaches, then qualitative two-dimensional risk levels are applied. Amland (01) and Wendland (12) map risk items to a 3x3 risk matrix with the two dimensions probability and impact with levels low, medium and high. The three cells at the lower left corner have low risk, the three cells at the upper right have high risk, and the remaining three cells have medium risk. Felderer et al. (11) apply a 2x2 risk matrix to determine the risk level and map their risk items with probability and impact values (each measured on a scale from 0 to 9) into this matrix. In the 2x2 risk matrix the lower left cell shows low risk, the upper right cell high risk, and the remaining two cells medium risk.

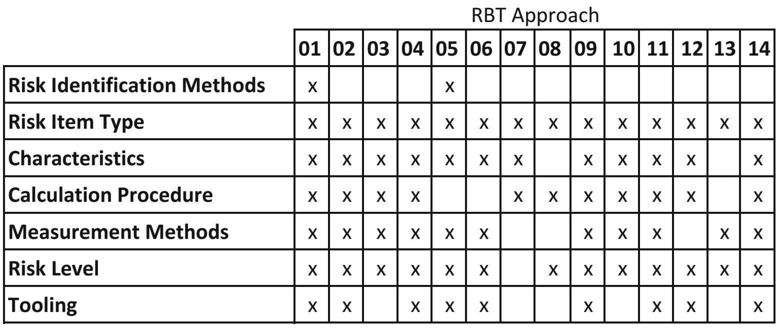
3.4 Tooling

If the risk assessment is not done ad-hoc, it requires tool support to be performed efficiently. The tooling may include printed forms as well as software tool support to perform the computations in an automatic way. Software tools supporting risk assessment for testing may be spreadsheets, specific risk assessment or management tools [28], or test or project management tools. The tooling is often fixed already before the risk model is defined, e.g., because a specific test management tool has to be used, and influences the definition of the risk model.

Amland (01) uses a spreadsheet for risk assessment. Souza et al. (05) use a specific risk assessment tool called RBTTool. Felderer et al. (11) use forms to conduct a risk assessment workshop. Finally, in Section 4 we integrate risk assessment into a project management tool.

Figure 3 summarizes which risk-based testing approach (RBT Approach) explicitly addresses which aspect of the risk assessment model. We skip the test scope as all listed RBT approaches implicitly presume that the prerequisites for the application of risk-based testing are fulfilled and defined.

The risk identification method is covered explicitly only be approaches 01 and 05. The risk item type is explicitly addressed by all RBT approaches. The remaining aspects are covered by most approaches. Only approach 01, i.e., Amland, covers all listed aspects explicitly.



	RBT Approach													
	01	02	03	04	05	06	07	08	09	10	11	12	13	14
Risk Identification Methods	x				x									
Risk Item Type	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Characteristics	x	x	x	x	x	x	x		x	x	x	x		x
Calculation Procedure	x	x	x	x			x	x	x	x	x	x		x
Measurement Methods	x	x	x	x	x	x			x	x	x		x	x
Risk Level	x	x	x	x	x	x		x	x	x	x	x	x	x
Tooling	x	x		x	x	x			x		x	x		x

Fig. 3. Elements of Risk Assessment Framework Covered by RBT Approaches

4 Application of Risk Assessment Model in an Industrial Test Process

In this section, we show how the risk assessment model is applied in the test process of a company in the telecommunication domain. The company follows a structured development and test process on the basis of a clearly-defined generic system and test model shown in Fig. 4. Further details on the company and its test process can be found in [29].

In this model, so called features are the central concept to plan and control implementation and testing. A *feature* has a concise and complete description of its functionality, along with non-functional aspects like performance or security. Features are on the one hand assigned to requirements and on the other hand to components. A *requirement* describes a certain functional or non-functional property of the system and is implemented by a set of features. A *component* is an installable artifact that provides the functionality of several features. Components are defined hierarchically in a tree. The root component represents the *system* and the leaves are *units*. As features are the tested artifacts, test cases are assigned to them. Differing from components, *testable objects* are executable units composed of one or more component and a test environment. Testable objects are assigned to the system or a component and have an attached test plan. A *test plan* contains test cases grouped either by features or components. Each test case contains a description, test steps and expected results.

Development and testing follow the V-model. First, a customer solution manager collects the requirements in a user requirements specification. Then, the features are defined and the system architecture is derived by a technical solution manager. The features are assigned to requirements in the technical requirements specification. The system design is then further refined to concrete components with assigned units, which are implemented and tested by a developer. As soon as feature definitions are available, test planning is started. First, testable objects are defined and a test plan, which is based on formerly defined requirements acceptance criteria. It contains test cases grouped either by features or components and has a test end criterion. In the test design phase, executable test cases are defined by testers according to the test plan. Test cases are also adopted from existing components or features. New or changed test cases are reviewed and corrected if necessary. After the respective testable object (including its test environment) has been implemented, the test cases are executed. Each *test run* contains a *test result* for each of its executed test cases. Depending on the test results, a problem ticket is created. As soon as the test end criterion is reached, a test report is provided.

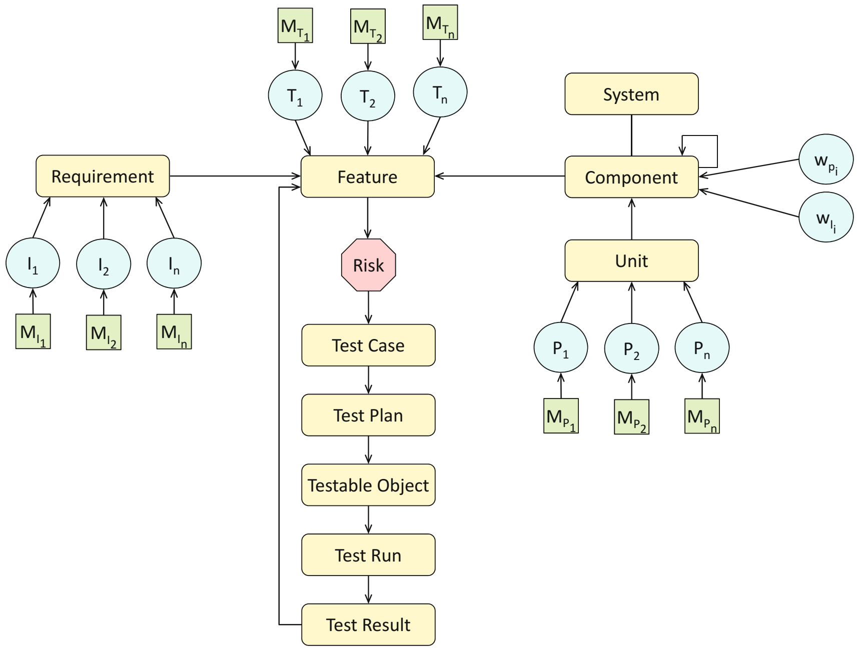


Fig. 4. System, Test and Risk Assessment Artifacts in Development and Test Process [29]

Test Scope. The expected benefits of risk-based testing are mainly decision support on resource allocation. The time resources for testing are limited as solutions have to be provided at fixed dates. Therefore, test cases should be prioritized for execution based on their risk level to mitigate highest risks in the limited test window. A test end criterion which considers the risk levels of features should terminate testing.

Risk Identification Methods. Due to the established development and test process, features were more or less already fixed as risk items. Risk identification put its focus on the identification of factors determining the risk assigned to features. For this purpose a list of factors is prepared from which suitable factors are selected by test managers in a separate workshop.

Risk Item Types. As test cases are linked to features, they are used as risk items to which risk exposure values are assigned as well. Features are traceable to requirements and components and therefore allow integrating a technical view on risk based on the components as well as a business view based on the requirements.

Characteristics. Risk is defined on the basis of the Factor-Criteria-Metrics model [27]. The definition considers the factors probability P and impact I as well as the additional factor time T. Probability reflects the technical view, impact the business view and time the system evolution. The factor values are determined by weighted criteria. The probability factor is composed of the weighted technical criteria code complexity, data complexity, functional complexity, visibility and third-party software. The impact factor is composed of the user and business-oriented weighted criteria usage, availability, importance and performance. The time factor is composed of the criteria bug tracking, change history, new technologies and project progress. Each factor is determined by a specific metric.

Measurement Methods. Probability, impact and time are explicitly defined on the basis of several criteria. For each criterion metrics are defined to determine the value in an objective way. The metrics can be determined manually by a suitable stakeholder or even automatically. For instance, the importance is measured manually on a five-point Likert scale and the code complexity is measured automatically by the McCabe complexity [30]. For the automatic measurement of source code metrics the source code quality management tool Sonar [31] is applied.

Calculation Procedure. Due to the traceability between requirements, components and units via features, the probability criteria are measured for units, the impact factors for requirements, and the weights are assigned to components (from which only a few exist). Time criteria are directly assigned to features, for which the risk exposure values are calculated (see Fig. 4). The probability P and the impact I are evaluated by several weighted criteria. For a risk item a, the probability P is for instance determined by the formula $P(a) = (\sum_{j=0}^{m} p_j \cdot w_j) \div (\sum_{j=0}^{m} w_j)$, where p_j are values for probability criteria and w_j are weight values for the criteria. The range of the criteria values are natural numbers between 0 and 9, and of the weights real numbers between 0 and 1 (so the weight can be naturally interpreted as scaling factor). The time factor, which scales the probability, has a range between 0 and 1, and is the mean of the time criteria values. The risk exposure value of a feature can be calculated via the formula, $R = (P \cdot T) \times I$, where P denotes the aggregated probability factor, I the aggregated impact factor, and T the aggregated time factor which reduces the value of P over time. Figure 6 shows the computed risk coefficients for seven features based values for the time criteria bug tracking, change history, new technologies and project progress. For each feature, the mean of the time criteria values is multiplied with the probability value and then combined with the impact value. For instance, all time criteria values of Feature 003 in Fig. 6 are 1. Therefore, the time factor T, i.e., the mean of the time criteria values, is 1 as well. With a probability factor P of 5 and an impact factor I of 6.75, the resulting risk coefficient R is $(5 \cdot 1) \times 6.75$, which corresponds to the value 33.75.

Risk Levels. The scaled probability and impact value defining the risk value are mapped to a 2x2 risk matrix to determine the risk level. Risk items, i.e., features with assigned risk values, mapped to the lower left cell have low risk, to the upper right cell high risk, and to the remaining two cells medium risk. Figure 5 shows the applied risk matrix. In this risk matrix, Feature 003 with risk coefficient $(5 \cdot 1) \times 6.75$ is of high risk.

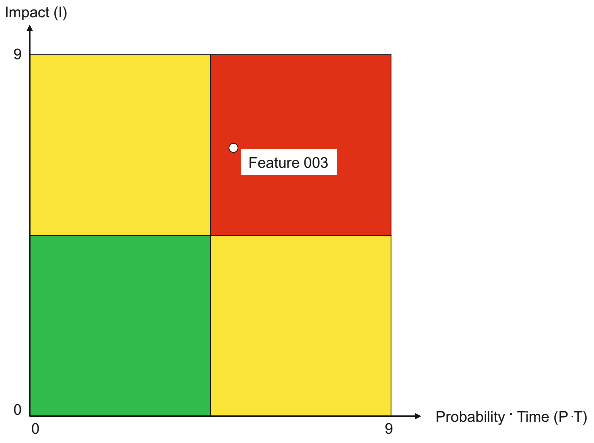
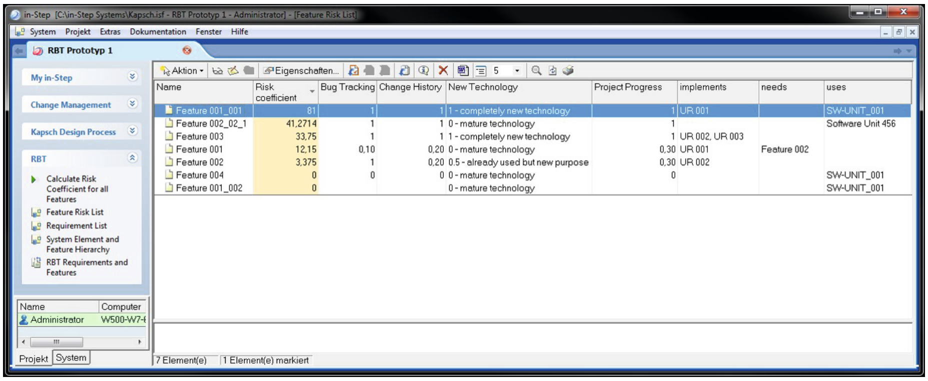


Fig. 5. Risk Matrix in Industrial Case

Tooling Risk assessment as well as the overall risk-based test process with all its artifacts and process steps is supported in the project management tool in-Step [32] which is already established in the company. Figure 6 shows a screenshot of the specifically developed risk assessment view of in-Step where the measures for criteria are entered and processed to calculate risk exposure values. In addition, for the automatic measurement of source code metrics the source code quality management tool Sonar [31] is used.

Risk-Based Test Process In the test process risks are explicitly considered in the test planning, test design and test execution phase. In the test planning phase, first testable objects, which are executable units composed of one or more components and a test environment, are defined. Then a test plan is created on the basis of formally defined test end criteria, features with attached risk exposure values and components. A typical test end criterion defines that all features with


in-Step [C:\in-Step Systems\Kapsch.isf - RBT Prototyp 1 - Administrator] - [Feature Risk List]
System Projekt Extras Dokumentation Fenster Hilfe
RBT Prototyp 1
My in-Step
Change Management
Kapsch Design Process
RBT
Calculate Risk Coefficient for all Features
Feature Risk List
Requirement List
System Element and Feature Hierarchy
RBT Requirements and Features
Aktion
Eigenschaften...
Projekt System
7 Element(e)
1 Element(e) markiert

Name	Computer
Administrator	W500-W7-(

Name	Risk coefficient	Bug Tracking	Change History	New Technology	Project Progress	implements	needs	uses
Feature 001_001	81	1	1	1 - completely new technology	1	UR 001		SW-UNIT_001
Feature 002_02_1	41,2714	1	1	0 - mature technology	1			Software Unit 456
Feature 003	33,75	1	1	1 - completely new technology	1	UR 002, UR 003		
Feature 001	12,15	0,10	0,20	0 - mature technology	0,30	UR 001	Feature 002	
Feature 002	3,375	1	0,20	0.5 - already used but new purpose	0,30	UR 002		
Feature 004	0	0	0	0 - mature technology	0			SW-UNIT_001
Feature 001_002	0			0 - mature technology				SW-UNIT_001

Fig. 6. Risk Assessment in Project Management Tool in-Step [29]

high risk have to be tested, features with medium risk are optional candidates to be tested in order to reach the required test coverage, and features with low risk are only tested if all others have been tested and resources are available. In the test design phase, executable test cases are defined by testers according to the test plan and get assigned the risk exposure value of the feature they are designed for. Already existing test cases which are applicable are selected from similar previous components or features. New or changed test cases are reviewed and corrected if necessary. After the respective testable object (including its test environment) is available for the test, the test cases are executed ordered by their risk level and risk exposure values (inherited from the assigned features). Each test run contains a test result for each of its executed test cases. Depending on the test results, a problem ticket is created. As soon as the test end criterion is reached, a test report is created. But the test report itself does not explicitly take risk information into account.

5 Summary and Future Work

In this paper we presented a risk assessment framework for testing purposes. The framework is based on the risk-based test process which is configured by and provides feedback for a risk assessment model. This model is the main component of our framework and defines the test scope, the risk identification method, a risk model as well as the tooling for risk assessment. The risk assessment framework is derived on the basis of best practices extracted from published risk-based testing approaches and applied to an industrial test process where it guides the definition and application of the RBT approach.

In future, we intend to provide a catalog with concrete guidelines on how to configure the risk assessment model to additionally support practitioners. In addition, we will perform empirical case studies to further evaluate and improve the risk assessment model.

Acknowledgment. This research was partially funded by the research projects MOBSTECO (FWF P 26194-N15) and QE LaB - Living Models for Open Systems (FFG 822740).

References

1. Felderer, M., Schieferdecker, I.: A taxonomy of risk-based testing. STTT (2014)
2. Wendland, M.F., Kranz, M., Schieferdecker, I.: A systematic approach to risk-based testing using risk-annotated requirements models. In: ICSEA 2012, The Seventh International Conference on Software Engineering Advances, pp. 636–642 (2012)
3. ISTQB: Standard glossary of terms used in software testing, version 2.2. Technical report, ISTQB (2012)
4. Alam, M.M., Khan, A.I.: Risk-based testing techniques: A perspective study. International Journal of Computer Applications 65(1) (2013)
5. ISO: ISO/IEC 29119 Software Testing, Draft (2013)
6. Amland, S.: Risk-based testing: Risk analysis fundamentals and metrics for software testing including a financial application case study. Journal of Systems and Software 53(3), 287–295 (2000)
7. Felderer, M., Haisjackl, C., Breu, R., Motz, J.: Integrating manual and automatic risk assessment for risk-based testing. In: Biffl, S., Winkler, D., Bergsmann, J. (eds.) SWQD 2012. LNBIP, vol. 94, pp. 159–180. Springer, Heidelberg (2012)
8. Merriam-Webster: Merriam-Webster Online Dictionary (2009), `http://www.merriam-webster.com/dictionary/risk` (accessed: April 04, 2013)
9. Standards Australia/New Zealand: Risk Management AS/NZS 4360:2004 (2004)
10. McCall, J., Richards, P., Walters, G.: Factors in software quality. Technical report, NTIS, vol. 1, 2 and 3 (1997)
11. Bai, X., Kenett, R.S., Yu, W.: Risk assessment and adaptive group testing of semantic web services. International Journal of Software Engineering and Knowledge Engineering 22(05), 595–620 (2012)
12. Felderer, M., Ramler, R.: Integrating risk-based testing in industrial test processes. Software Quality Journal, 1–33 (2013) (online first)
13. Ray, M., Mohapatra, D.P.: Risk analysis: a guiding force in the improvement of testing. IET Software 7(1), 29–46 (2013)
14. Chen, Y., Probert, R.L., Sims, D.P.: Specification-based regression test selection with risk analysis. In: Proceedings of the 2002 Conference of the Centre for Advanced Studies on Collaborative Research, p. 1. IBM Press (2002)
15. Redmill, F.: Exploring risk-based testing and its implications. Software Testing, Verification and Reliability 14(1), 3–15 (2004)
16. Redmill, F.: Theory and practice of risk-based testing. Software Testing, Verification and Reliability 15(1), 3–20 (2005)
17. Stallbaum, H., Metzger, A.: Employing requirements metrics for automating early risk assessment. In: Proc. of MeReP 2007, Palma de Mallorca, Spain, pp. 1–12 (2007)
18. Stallbaum, H., Metzger, A., Pohl, K.: An automated technique for risk-based test case generation and prioritization. In: Proceedings of the 3rd International Workshop on Automation of Software Test, pp. 67–70. ACM (2008)
19. Souza, E., Gusmao, C., Alves, K., Venancio, J., Melo, R.: Measurement and control for risk-based test cases and activities. In: 10th Latin American Test Workshop, pp. 1–6. IEEE (2009)

20. Souza, E., Gusmão, C., Venâncio, J.: Risk-based testing: A case study. In: 2010 Seventh International Conference on Information Technology: New Generations (ITNG), pp. 1032–1037. IEEE (2010)
21. Zimmermann, F., Eschbach, R., Kloos, J., Bauer, T., et al.: Risk-based statistical testing: A refinement-based approach to the reliability analysis of safety-critical systems. In: Proceedings of the 12th European Workshop on Dependable Computing, EWDC 2009 (2009)
22. Kloos, J., Hussain, T., Eschbach, R.: Risk-based testing of safety-critical embedded systems driven by fault tree analysis. In: 2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops (ICSTW), pp. 26–33. IEEE (2011)
23. Yoon, H., Choi, B.: A test case prioritization based on degree of risk exposure and its empirical study. International Journal of Software Engineering and Knowledge Engineering 21(02), 191–209 (2011)
24. Zech, P.: Risk-based security testing in cloud computing environments. In: 2011 IEEE Fourth International Conference on Software Testing, Verification and Validation (ICST), pp. 411–414. IEEE (2011)
25. Felderer, M., Ramler, R.: Experiences and challenges of introducing risk-based testing in an industrial project. In: Winkler, D., Biffl, S., Bergsmann, J. (eds.) SWQD 2013. LNBIP, vol. 133, pp. 10–29. Springer, Heidelberg (2013)
26. Pandian, C.R.: Applied software risk management: a guide for software project managers. CRC Press (2006)
27. Cavano, J., McCall, J.: A framework for the measurement of software quality. ACM SIGMETRICS Performance Evaluation Review 7(3-4), 133–139 (1978)
28. Haisjackl, C., Felderer, M., Breu, R.: Riscal–a risk estimation tool for software engineering purposes. In: 2013 39th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), pp. 292–299. IEEE (2013)
29. Felderer, M., Ramler, R.: A multiple case study on risk-based testing in industry. STTT (2014)
30. McCabe, T.: A complexity measure. IEEE Transactions on Software Engineering, 308–320 (1976)
31. SonarSource: Sonar (2013), `http://www.sonarsource.org/` (accessed: March 12, 2013)
32. microtool: in-Step (2013), `http://www.microtool.de/inStep` (accessed: November 30, 2013)

Data Driven Testing of Open Source Software

Inbal Yahav, Ron S. Kenett, and Xiaoying Bai

Graduate School of Business Administration, Bar Ilan University
Inbal.yahav@biu.ac.il
The KPA Group, Israel, Univ. of Torino, Italy and NYU-Poly, NY, USA
ron@kpa-group.com
Dept. of Comp. Science and Technology, Tsinghua University, China
baixy@tsinghua.edu.cn

Abstract. The increasing adoption of open source software (OSS) components in software systems introduces new quality risks and testing challenges. OSS components are developed and maintained by open communities and the fluctuation of community members and structures can result in instability of the software quality. Hence, an investigation is necessary to analyze the impact open community dynamics and the quality of the OSS, such as the level and trends in internal communications and content distribution. The analysis results provide inputs to drive selective testing for effective validation and verification of OSS components. The paper suggests an approach for monitoring community dynamics continuously, including communications like email and blogs, and repositories of bugs and fixes. Detection of patterns in the monitored behavior such as changes in traffic levels within and across clusters can be used in turn to drive testing efforts. Our proposal is demonstrated in the case of the XWiki OSS, a Java-based environment that allows for the storing of structured data and the execution of server side scripts within the wiki interface. We illustrate our concepts, methods and approach behind this approach for risk based testing of OSS.

1 Introduction

Open Source Software (OSS) is playing a leading role in current information technology practices. Its pervasive adoption is not without risks for an industry that has experienced significant failures in product quality, timelines and delivery costs. Inadequate testing and risk management has been identified among the top deficiencies when implementing OSS-based solutions. A crucial aspect in managing and mitigating OSS adoption risks is that of deeply understanding the behavior and dynamics of the OSS communities that provide the software components. This can be achieved via the analysis of big amounts of data related, for example, to community activeness, reliability, or capacity of managing the OSS component maintenance and evolution. In this work we combine and understanding of OSS community dynamics with advanced analytic methods and an underlying approach of risk based testing to address the issues listed above. Our goal is to provide OSS adopters and OSS integrators with an approach for designing tests in a focused and adaptable way.

The paper begins with a section providing background on OSS testing and a review of related work. Section 3 provides a comprehensive presentation of analytics for capturing community and communication data from OSS communities. Section 4 introduces the XWiki case study used to demonstrate the proposed approach and section 5 provides numerical results in the context of this case study. A final section concludes with a highlight of our proposal and directions for future work.

2 Background and Related Work

Software testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test. Software testing can also provide an objective, independent view of the software quality and performance that allows the business stakeholders to assess and understand the quality of software implementation. Test techniques include executing a program or application with the intent of finding software bugs.

The software testing activity consists of processes for validating and verifying that a computer program/application/product meets the requirements that guided its design and development, works as expected, can be implemented with the same characteristics, and satisfies the needs of stakeholders.

Traditionally, most of the test effort occurs after the requirements have been defined and the coding process has been completed. In the Agile approaches most of the test effort is on-going and development is incremental. In general, the testing methodology is governed by the chosen software development methodology [1].

In most cases, tests are performed by test engineers. Testers can be part of the development organization, part of the user's organization or operate as an independent entity. The advantages of an independent functional test team are that test team resources are efficiently distributed and can be easily reassigned to different products, test engineers are working according to a proven testing methodology, which gives them the ability to test faster, better and at lower costs, knowledge and experience is accumulated across many projects, is formalized and shared, the complexities of testing issues are handled professionally until they are fixed and verified and verification of the software is done professionally, providing sufficient coverage of the new code. Testing can never completely identify all the defects within software. Instead, it furnishes a criticism or comparison that compares the state and behavior of the product against documents such as requirements documents or design documents. A primary purpose of testing is to detect software failures so that defects may be discovered and corrected, before full scale operational deployment.

Testing is inherent in a wide range of software applications. For an example of a risk based methodology for group testing of web services see [3]. Another type of testing is focused on usability of software applications where analytic methods like Bayesian networks and Markov models are used to analyze weblogs [4,5]. Some early attempts have been published where analytic methods are used to analyze characteristics of software failures [1, 7, 8]. In this work we use analytics to correlate dynamics of open source communities with reported bugs. A related work focused on risk management and business risks is presented in [2].

In open source software (OSS) development, the identification of factors affecting software quality is critical for the design of an effective testing strategy.

Open Source Software (OSS) is supplied by an OSS community, can be produced in-house; can be part of the organization's value proposition, can be used as infrastructure for the development of software or for the execution of business processes; can be sold in order to allow for revenues; can determine customer segments or can be used in order to lower costs.

Several European research projects considered the problem of evaluating the qualities and characteristics of OSS software. FLOSSMetrics (http://flossmetrics.org/) objective was to construct, publish and analyze a large scale database with information and metrics about OSS development coming from several thousand software projects, using existing methodologies, and tools already developed. The objective of the QualiPSO project (http://qualipso.org) was to improve the quality of OSS projects, in particular focusing on its maturity. It provides a maturity model, similar to CMMI (Kenet and Baker, 2010), which is composed of a list of topics to be addressed by a project in order to be categorized in one of the three levels of maturity defined. The project also produced several tools that help to analyze the source code, the bug tracking systems, and a platform to integrate these tools. The QualOSS project (http://www.qualoss.eu/) defined a method to assess the quality of OSS projects, more concretely, their qualities of robustness and evolvability. To do so, it designed a quality model, and described a process for the quality assessment. In this case, the assessment is made manually by means of long checklists. It also provides a tool to facilitate the process. The quality model of QualOSS is composed of three types of interrelated elements: quality characteristics, concrete attributes of a product or community, metrics (about 150), concrete aspects that can be measured, and indicators, that define how to aggregate and evaluate the measurement values to obtain consolidated information. OSSMETER (http://www.ossmeter.eu) aims to develop a platform that supports decision makers in the process of discovering, comparing, assessing and monitoring the health, quality, impact and activity of open-source software. To achieve this, OSSMETER computes trustworthy quality indicators by performing advanced analysis and integration of information from diverse sources including the project metadata, source code repositories, communication channels and bug tracking systems of OSS projects. The RISCOSS project (www.riscoss.eu) develops a platform is to monitor and flag changes to measurable properties of open source artifacts that are indicative of the occurrence of potential business risks to adopting organizations. RISCOSS is based on an organizational mapping using the i* methodology and combines disjunctive logic reasoning with statistical analysis for risk monitoring and risk mitigation management [2].

3 Open Source Software Assessment Based on Community and Communication Data

In this work we focus on analyzing data from OSS communities to drive testing activities of OSS adopters. We use throughout examples from the XWiki community introduced in section 3 which is also part of RISCOSS.

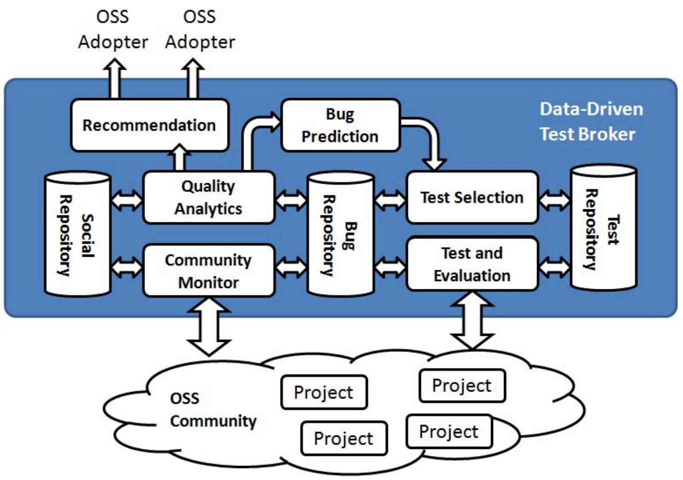


Fig. 1. Approach Overview

Figure 1 gives an overview of the proposed approach. The data-driven testing system (DDT) serves as a broker between OSS adopters and OSS communities. For a OSS project, the adopters always concerns various quality issues including both software issues (such as reliability, performance, and interoperability) and community issues (such as maturity, expertise, and stability). DDT aims to provide recommendations to OSS adopters in response to their quality queries, by continuous monitoring and evaluation of projects in OSS communities.

DDT basically contains two parts: quality centered data analytics, and data-driven test selection. OSS projects are usually developed and maintained by open communities. The fluctuation of community members and structures can results in instability of the software quality. For example, voluntary developers may join and quit the community flexibly. There is no guarantee of consistent quality between different versions, code branches, and development groups. Hence, an investigation is necessary to analyze the correlation between the social aspects of open communities and the quality of the OSS, such as the dynamics in communications and content distribution. The traceability ensures that whenever a change is detected, a regression process will be triggered automatically to evaluate quality risks in terms of bug probability.

The analysis results can be taken as inputs to drive selective testing for effective validation and verification of OSS projects. As exhaustive testing is infeasible in most cases due to time and resources limitations, selective testing techniques are usually needed to allocate test resources to the most critical components and features. Risk-based testing is a technique to schedule tests according to the risk measurements on the object under tests. In this research, the monitored community dynamics are used to measure the risk of an OSS component to guide the component's testing activities.

As shown in Figure 1, three repositories are built to support the process. Social repository stores the social network data for the community. Bug repository is the core connection between community behavior and project quality. It will trace bug reports in the community from various aspects, such as by project, by version, by code branches, by modules, by developer, by different role and group. Test repository traces test scripts to OSS projects.

We focus here on insights for OSS test strategy design generated by a two level analysis. First, we investigate the relationship between the social network of the XWiki communities and the timing and frequency of bug reports. Specifically, we examine whether software projects with more bug reports have a stronger (weaker) developer community, and whether software projects with more bug reports have more frequent (less frequent) email communications.

Secondly, we examine whether the developers' communications and communities can be used to predict bug reports, and consecutively, used within a mechanism for dynamic test selection. Such an early bug predictor can provide an effective test selection and initiation strategy. The next section introduces the XWiki community.

4 The XWiki Case Study

XWiki (http://www.xwiki.com) is a Java-based environment that allows for the storing of structured data and the execution of server side script within the wiki interface. XWiki was originally written by Ludovic Dubost in 2003. In 2006, the Apache Maven developer, Vincent Massol became the lead developer of the OOS XWiki. In this work, we use XWiki data to demonstrate our data driven testing approach. Specifically consider an adopter of XWiki and related contributed projects. The adopter has integrated these OSS components in a proprietary system and is designing a test strategy that covers XWiki functionality in terms of its contributed OSS projects. Given the dynamics of the OSS community and the evolution of the XWiki components, the test strategy needs to be updated and adapted. The approach introduced here is data driven and based on online analytics. The data used here consists of: mailing lists archives of users and developers, IRC chat archives, commits via git, code review comments, and information about bugs and releases. To illustrate our method we focus on users and developers email communications. Emails are available since 2005 to 2013, a total of nearly 60,000 massages (including original messages and replies). We generate the social network of communication, analyze the community of members, and monitor their within and across community massages exchange.

Email data, by nature, is very noisy. Users can use different accounts to post and reply messages. Content exchanged is free text, and as such contains typos, synonyms, plurals, and more. A massive data pre-processing step is required in order to generate an informative social network. In this work, we use a name-scheme matching approach to detect multiple user's accounts. In specific, we use full and partial texts comparisons to find plausible users matches and their probability. We then score our results with manual intervention.

Content analysis is based on keywords extraction from XWiki's list of projects (http://jira.xwiki.org/secure/BrowseProjects.jspa#all). Mentions of projects in an email communication (title only) are counted. Email communications are then classified to zero or more projects. For example, the text 'LDAP authentication not working' is classified as related to the project "LDAP".

Information regarding bugs reports is also available for XWiki projects (http://jira.xwiki.org/browse/XWS-220?jql=). We use information about bugs as an indication of "missed test", and focus on developing a mechanism that select and run test cases before bugs are being reported.

In this research we use data from Emails between users and lists of reported bugs in XWiki contributed projects (see Figure 2).

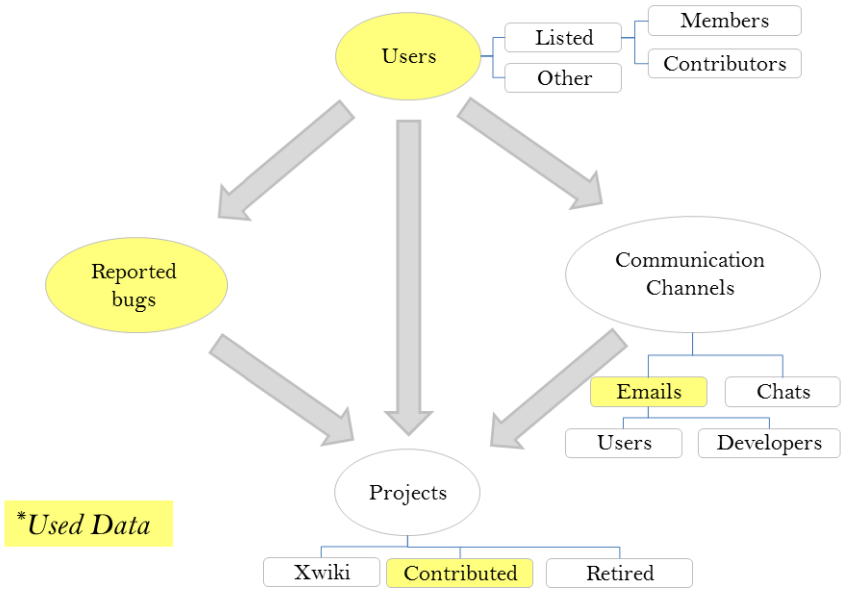


Fig. 2. Sources of data in XWiki case study

5 Numerical Analysis

5.1 OSS Community Analytics: Data Preprocessing

In general, three types of communities make up the social network of an OSS community: (1) team members, (2) developers (contributors) and expert users, and (3) users. The official roles of XWiki team members is available on the XWiki website (www.xwiki.com/lang/en/Company/Team).

To detect the developers' and expert users' communities we use Clauset-Newman-Moore community detection algorithm [6]. As input to the algorithm, we use the sub-social network of developers only, disregarding members' and users' network. The communities that were detected by the algorithm are given in Figure 3.

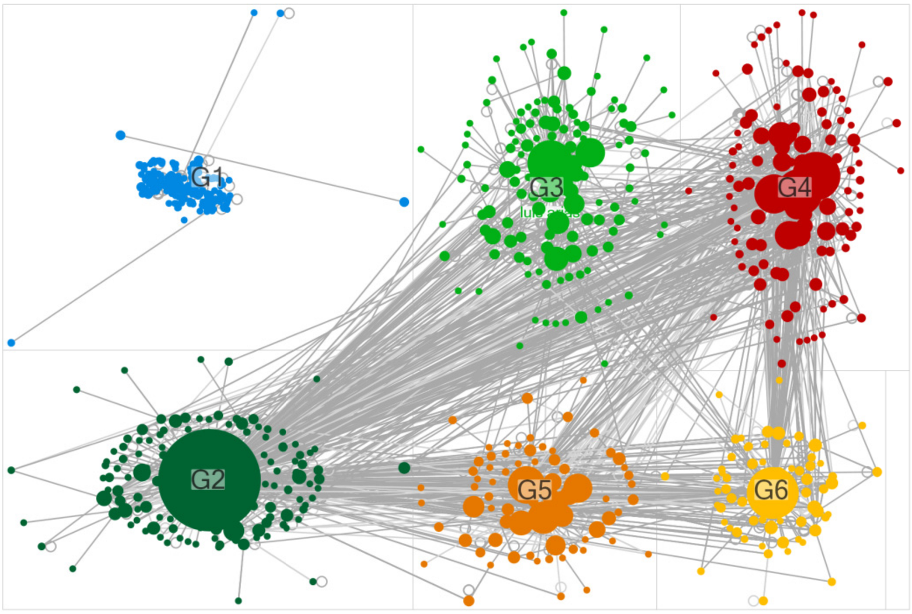


Fig. 3. Developers' communities

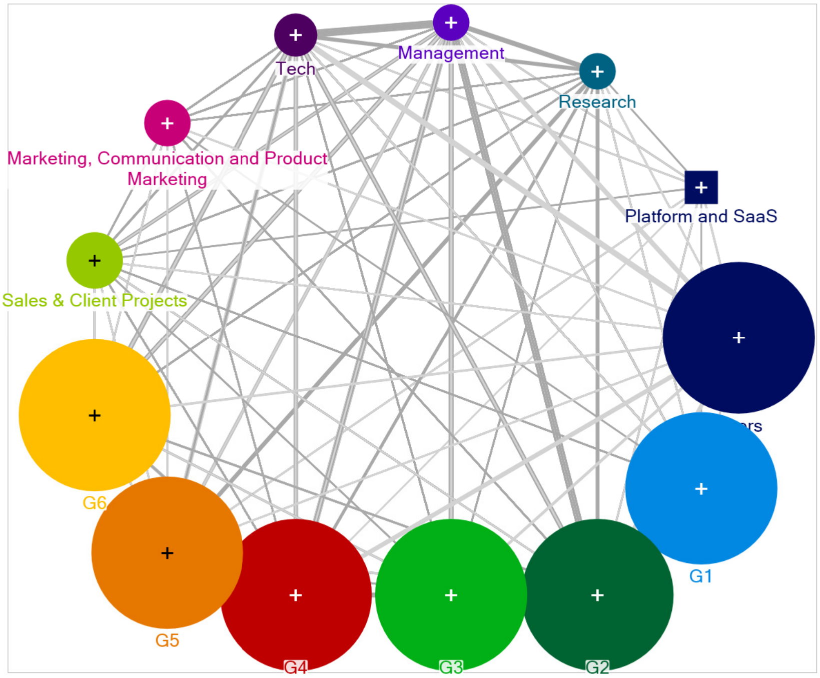


Fig. 4. Communities of XWiki OSS

The figure depicts the directed graph of developers' email communications, clustered into 6 communities (G1 through G6). Node size corresponds to centrality in the *entire* network (including team members and users). Dark arcs in the network correspond to communication via the developers' email channel. Light arcs correspond to communication via the users' email channel.

From the 6 communities detected, the first community (G1) has no inter-community communication, or across developers' communities. The G1 cluster, therefore gathers small developers or expert users that communicate with either users or members of the OSS team. The other 5 communities (G2 through G6) communicate both within the community and across the developers' communities.

Communication across the communities is plotted in Figure 4.

This rendering and analysis of communication dynamics is used as data preprocessing in the insight generation process described above.

5.2 Bugs and Social Networks: First Level Analytics

Following the social analysis data preprocessing, we study the relationship between the social network of OSS community and reported bug. At first, we ignore the timeliness dimension and the communities' dynamics of the problem, and examine a snapshot of the entire data at aggregation. The rationale behind predicting bugs is as follows: if we can predict bugs in a specific component before the actual report, we can use this information to automatically run test cases related to this component, and fix the bug before it affects the users. In other words, bugs prediction is used as a proxy for required test cases.

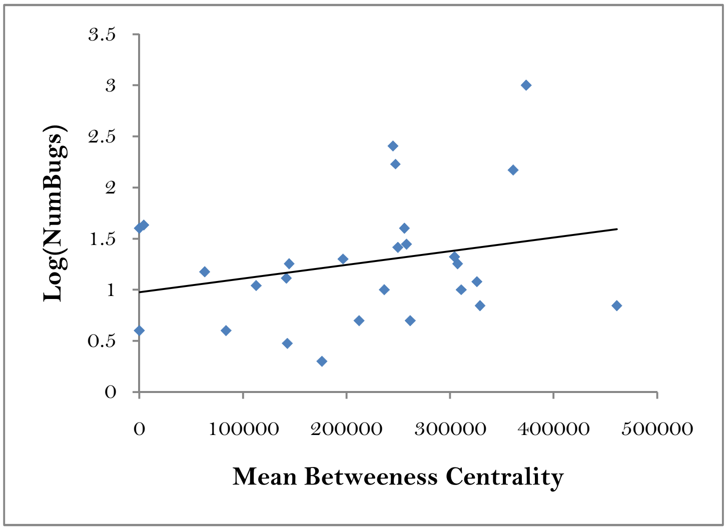


Fig. 5. The relationship between bugs and centrality of developers that talk about the project

Figure 5 depicts the relationship between reported bugs per project and the centrality of the people who discuss them. It shows that the more central are the people that talk of a project, the larger is the number of bugs reported in the project. There are two plausible explanations to this relationship: (1) risky projects “attract” central developers, or (2) experts users and developers find more bugs (in other words, if a project is used by central, frequent users, it is more likely that its bugs will be detected). In the first explanation, bugs drive communication. In the second explanation, use drives communication that drives bug detection. A combination of the two is also likely.

Next, we add the time dimension to our analysis. Figure 6 illustrates the relation between cumulative communication over time and the cumulative number of reported bugs over time, for a selected list of contributed projects. The relationship between communications and reported bugs is not consistent for all project, although it is definitely clear for some.

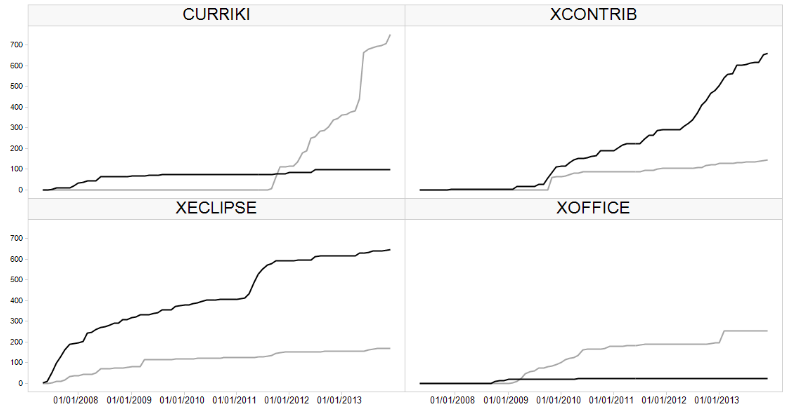


Fig. 6. Total communications related to several projects vs. reported bugs over time. Gray: bug reports; Black: email communications.

Projects XECLIPSE and XCONTRIB show a similar patter linking bug reports and email communication. A possible explanation for the low and saturated number of reported bugs in project XOFFICE is bug under-reporting. Project CURRIKI seems to behave differently from XECLIPSE and XCONTRIB. Our goal here, however, is to identify a good predictor of the occurrence of bugs that is calculated from the email communication levels within the OSS community. Such prediction can be used to drive the test selection and activation process as presented in section 2.

5.3 Test Case Selection: : Second Level Analytics

In this section we develop a prediction model that predicts whether a bug will be reported on a given project, in a certain day. This information can be used as the basis of efficient test selection.

We model a bug occurrence indicator at time *t*, for project *p,* with a logistic regression (denoted *f)* of the email communications in different communities (*c*), and the time of the last reported bug. Equation 1 presents the conceptual prediction model.

$$bug_t^p = f\left(\overline{Communication_{t-1}^p(c)}, time\ since\ last\ bug^p\right) \qquad (1)$$

To avoid model over-fitting, we consider projects with over 20 reported bugs, and mentioned over 20 times in email communications. That leaves us with a subset of 4 contributed projects, namely CURRIKI, XCONTRIB, XECLIPSE, and XOFFICE.

In a first analysis, we fit a single model to all 4 projects. The resulted logistic coefficients are given in Table 1. Coefficients of communities' communications are highlighted in gray. Several communities' communications, such as that of G2, G3, Research group and others, appear as very informative in predicting reporting of bugs.

Table 1. Logistic regression for predicting bug indicators

Coefficient	Estimate	p-value
(Intercept)	-0.69	0.00
Project XCONTRIB	-0.96	0.03
Proj XECLIPSE	-0.40	0.34
Proj XOFFICE	-0.01	0.98
Community G1	-0.43	0.72
Community G2	**0.26**	**0.12**
Community G3	**0.25**	**0.13**
Community G4	0.04	0.70
Community G5	**0.12**	**0.20**
Community G6	**-0.20**	**0.15**
Community Management	-0.10	0.31
Community Marketing, Communication and Product Marketing	0.42	0.30
Community Platform and SaaS	15.44	0.98
Community Research	**0.39**	**0.03**
Community Sales and Client Projects	**1.66**	**0.01**
Community Tech Support	0.07	0.56
Community Users	0.02	0.90
time Since Last Bug	0.00	0.45

The overall performance of the model is evaluated by a lift curve shown in Figure 7. The gray lift represents a performance of a model that does not use communications as predictors. Clearly, its performance is poor (very close to random). The black lift represents the performance of the model described above, which does use email communications as predictors. The delta AUC (Area Under the Curve) of the two models, is the contribution of our approach to the problem of efficient test case selection.

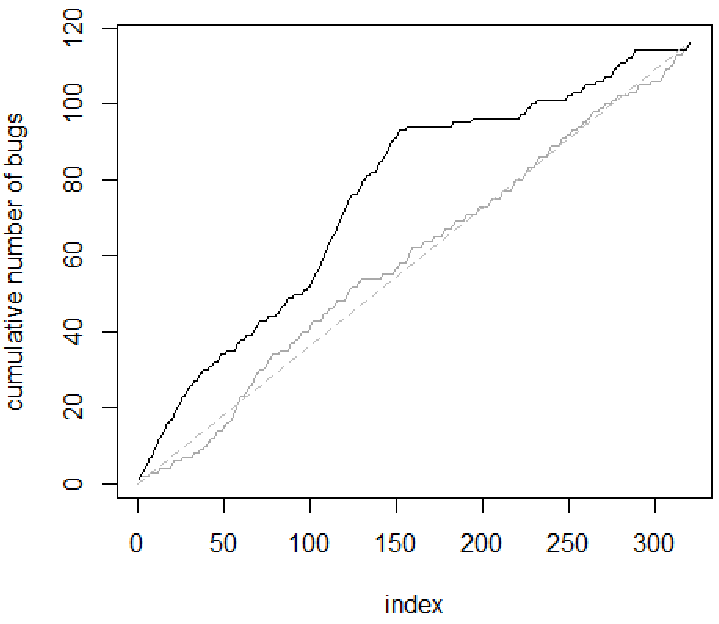


Fig. 7. Lift curve of the logistic regression with communication (black) and without (grey)

We next summarize the prediction performance in a confusion matrix (we use a default cutoff value on the probability of 0.5)

Table 2. Confution matrix of the logistic regression

Actual / Predicted	Predicted no-bug	Predicted bug
No-bug	77	127
Bug	19	97

In a second analysis, we generate a separate logistic model for each project. The separation of the models provides more flexibility in estimation of the communities' coefficient. This implies that different communities can be informative for predicting bugs in different projects. In Table 3 we list the most informative communities for each project. As expected, the list differs from one project to another. Interestingly, there is no benefit to our social networks clusters when predicting *reported* bugs for project XOFFICE. Again, the reason might be a possible under-reporting phenomenon in this project.

Table 3. Logistic regression per project: Informative communities

Project	Most informative communities
CURRIKI	Management (p-value = 0.31)
XCONTRIB	G6 (p-value = 0.14), Management (p-value = 0.14), Sales and Client Projects (p-value = 0.24), G2 (p-value = 0.25)
XECLIPSE	G2 (p-value = 0.02) G6 (p-value = 0.31)
XOFFICE	None

Confusion matrixes of our prediction model, per project, are given in Tables 4 through 7. The results show a very high detection level, with relatively low false detection, for most projects.

Table 4. Confusion matrix for project CURRIKI

Actual / Predicted	Predicted no-bug	Predicted bug
No-bug	53	0
Bug	1	26

Table 5. Confusion matrix for project XCONTRIB

Actual / Predicted	Predicted no-bug	Predicted bug
No-bug	43	10
Bug	3	24

Table 6. Confusion matrix for project XECLIPSE

Actual / Predicted	Predicted no-bug	Predicted bug
No-bug	37	9
Bug	11	23

Table 7. Confusion matrix for project XOFFICE

Actual / Predicted	Predicted no-bug	Predicted bug
No-bug	40	12
Bug	5	23

6 Summary and Conclusions

The objective of this study is to evaluate the feasibility of driving test efforts of OSS components on the basis OSS community data. The goal is to predict occurrence of bugs using email communication traffic and social network dynamics with data from an OSS community. The approach bears some similarity to customer targeting efforts by commercial banks who want to identify soon to be home buyers or Facebook that attempts to identify high school students applying for college in order to direct promotional campaigns. Here we want to prioritize testing effort. The open access of web data and advanced analytics, such as social network analysis, provide new opportunities in such rational test effort design. The amount and granularity of the available data determines the level of detail of the testing strategy implementation. With enough data, one can pinpoint bugs of specific types and locations.

In this work we mainly focused on data stratified by projects contributed to the XWiki platform. More in depth studies are required to map bugs to specific features and trigger such focused tests. Future research will also involve a wide breath approach. Typically OSS-based solutions are not developed in isolation. Instead, they exist in the wider context of an organization or a community, in larger OSS-based software ecosystems, which include groups of projects that are developed and co-evolve within the same environment, but also further and beyond their context, including the organization itself. Testing such ecosystems provides added complexities that require innovative solutions. We believe that properly combining analytic tools, with an in depth study of OSS community dynamics can lead to improvements in dealing with these issues.

Acknowledgement. The work of RSK was partially funded by the RISCOSS project under the EC 7th Framework Programme FP7/2007-2013 under the agreement number 318249.

References

1. Kenett, R.S., Baker, E.: Process Improvement and CMMI for Systems and Software. CRC Press (2010)
2. Franch, X., Susi, A., Annosi, M.C., Ayala, C., Glott, R., Gross, D., Kenett, R., Mancinelli, F., Pop Ramsamy, C.T., Ameller, D., et al.: Managing risk in open source software adoption. In: Proc. 8th Int. Conf. on Software Engineering and Applications (ICSOFT-EA 2013). SciTePress (2013)
3. Bai, X., Kenett, R.S., Yu, W.: Risk assessment and adaptive group testing of semantic web services. International Journal of Software Engineering and Knowledge Engineering 22(05), 595–620 (2012)
4. Harel, A., Kenett, R.S., Ruggeri, F.: Modeling web usability diagnostics on the basis of usage statistics. In: Statistical Methods in eCommerce Research, pp. 131–172 (2008)
5. Kenett, R.S., Harel, A., Ruggeri, F.: Controlling the usability of web services. International Journal of Software Engineering and Knowledge Engineering 19(05), 627–651 (2009)
6. Clauset, A., Newman, M.E., Moore, C.: Finding community structure in very large networks. Physical Review E 70(6), 066111 (2004)
7. Nagappan, N., Ball, T., Zeller, A.: Mining metrics to predict component failures. In: Proceedings of the 28th International Conference on Software Engineering (ICSE 2006), pp. 452–461. ACM, New York (2006)
8. Hata, H., Mizuno, O., Kikuno, T.: Bug prediction based on fine-grained module histories. In: Proceedings of the 2012 International Conference on Software Engineering (ICSE 2012), pp. 200–210. IEEE Press, Piscataway (2012)
9. Kim, S., Whitehead Jr., E.J., Zhang, Y.: Classifying Software Changes: Clean or Buggy? IEEE Trans. Softw. Eng. 34(2), 181–196 (2008)

Combining Risk Analysis and Security Testing[*]

Jürgen Großmann, Martin Schneider,
Johannes Viehmann, and Marc-Florian Wendland

Fraunhofer FOKUS,
Kaiserin-Augusta-Allee 31,
D-10589 Berlin, Germany
{Juergen.Grossmann,Martin.Schneider,Johannes.Viehmann,
Marc-Florian.Wendland}@Fokus.Fraunhofer.de

Abstract. A systematic integration of risk analysis and security testing allows for optimizing the test process as well as the risk assessment itself. The result of the risk assessment, i.e. the identified vulnerabilities, threat scenarios and unwanted incidents, can be used to guide the test identification and may complement requirements engineering results with systematic information concerning the threats and vulnerabilities of a system and their probabilities and consequences. This information can be used to weight threat scenarios and thus help identifying the ones that need to be treated and tested more carefully. On the other side, risk-based testing approaches can help to optimize the risk assessment itself by gaining empirical knowledge on the existence of vulnerabilities, the applicability and consequences of threat scenarios and the quality of countermeasures. This paper outlines a tool-based approach for risk-based security testing that combines the notion of risk-assessment with a pattern-based approach for automatic test generation relying on test directives and strategies and shows how results from the testing are systematically fed back into the risk assessment.

Keywords: Risk assessment, security testing, test pattern.

1 Introduction

Security is crucial in various market sectors, including IT, health, aviation and aerospace. In the real world, perfect security often cannot be achieved. Trust allows human beings to take remaining risks. Before trusting, before taking risks, it is reasonable to carefully analyze the chances, the potential benefits and the potential losses as far as possible. For technical systems, services and applications such an analysis might include risk assessment and security testing.

Those offering security critical technical systems, applications or services can benefit from careful risk analysis and security testing in two ways: They can use the

[*] The research leading to these results has also received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreements no 316853 and no 318786.

results to detect and treat potential weaknesses in their products. Additionally, they can use the results to communicate the identified remaining risks honestly, which can be very important to create trust.

This paper introduces new concepts to integrate compositional risk assessment and security testing into a single process. Furthermore, ideas for increasing the reusability of the risk analysis and security testing artifacts are presented. A focal point is laid onto the systematic integration of risk information and the test design process. In this paper, we present an approach to model-based risk-driven test design. We are going to employ so called test models, a formal specification of test design techniques (as part of the model) as well as model-based representation of risk information in order to actually generate test artifacts (such as test cases and test data) based on risk.

Implementing the described methodology in a tool in order to make it practically applicable for large scale systems for which manual analysis is not practicable is currently ongoing work.

2 The Problems

There is little doubt that security critical technical systems should be carefully analyzed. However, both risk assessment and security testing might be difficult and expensive. Each activity on its own requires experts. The systematic integration of both is a non-trivial challenge, especially in industrial projects where testers are not necessarily experts in security testing and risk analysis.

Typically, risk assessment is performed at a high level of abstraction and results depend on the experience and on subjective judgment of the analysts. Hence, results might be imprecise, unreliable and uncertain.

In contrast to risk assessment, security testing does produce objective and precise results – but only for those things that are actually tested. Even for small systems, complete testing is usually not possible since it would take too long and it would be by far too expensive. The selection of relevant test cases is a critical decision. Even highly insecure system can produce lots of correct test verdicts if the “wrong” test cases have been created and executed. Thus, the selection of appropriate test cases needs to be carried out in a systematic and comprehensible manner. Risk-based testing in general is a means to justify why a certain test case or test datum has been designed and executed in the first place. According to Bach [15], risk-based testing aims at testing the right things of a system at the right time. The idea of risk-based testing is simple: Identify prior to test case design those scenarios that provoke the most critical situations for a system to be tested and ensure that these critical situations are both effectively mitigated and sufficiently tested.

Additionally, test results can be systematically integrated with the risk assessment results in order to verify whether the assumed criticality of the system to be tested actually corresponds to the actual implementation of the system. By doing so, the risk analysis results can be refined with respect to the actual test results. If the tests pass, the security properties are met and the integration of the security measures seem to be properly realized. Thus, we have gained confidence that at least for the tested situations the system is secure. If a test has failed, thus, if an unwanted incident occurred, we gained confidence that the countermeasures are insufficient or could be circumvented for at least one case. This might require additional counter measures to be specified and implemented or deficient countermeasure implementations to be fixed. The main problem is that there is a lack of reliable and applicable methodologies on how to integrate the various pieces of information relevant for a risk-based testing or test-based risk assessment approach in a systematic way.

3 State of the Art

Risk assessment means to identify, analyze and evaluate risks, which threaten assets [1] [2]. There are lots of different methods and technologies established for risk assessment, including Fault Tree Analysis (FTA) [4], Event Tree Analysis (ETA) [5], Failure Mode Effect (and Criticality) Analysis (FMEA/FMECA) [3] and the CORAS method [6]. However, most traditional risk assessment technologies analyze systems as a whole [7]. They do not offer support for compositional risk assessment. Compositional risk assessment combines risk analysis results for components of a complex modular system to derive a risk picture for the entire complex system without looking further into the details of its components. Nevertheless, for the mentioned risk assessment concepts, there are some publications dealing with compositional risk analysis, e.g. [8] for FTA and [9] for FMEA.

While traditional testing tries to test specified functionality, security testing aims for identifying weaknesses and vulnerabilities to uncover unwanted behavior.

Currently, there are basically two different ways how security testing and security risk analysis can be combined [10]. Test Based Security Risk Assessment (TBRA) tries to improve the security risk analysis with the help of security risk testing and the final output results are risk analysis artifacts. There have been several publications about this approach, e.g. [11] [12], but there is no general applicable methodology and not much tool support.

In contrast to Test Based Security Risk Assessment, the Risk Based Security Testing (RBST) approach tries to improve security testing with the help of security risk analysis and the final results are test result reports. There are lots of different methods, some trying to identify test cases while others try to prioritize test cases or to do both. For example, Kloos et al. uses fault trees as the starting point for identifying test cases [13]. Stallbaum and Metzger automated the generation of risk-based test suites based on previously calculated requirements metrics [16] [17]. A prototype research tool called RiteDAP has been presented as being able to generate test cases out of weighted activity diagrams. Basically, it ranks paths in the activity diagram due to the risk they include.

Bauer and Zimmermann have presented a methodology called *sequence-based specification* to express formal requirements as low-level mealy machines for embedded safety-critical systems (e.g., [18] [19]). They build a system model based on the requirements specification. Afterwards, the outcome of a hazard analysis is weaved into the mealy machine. The correctness of the natural language requirements is actually assumed to hold. Finally, they describe an algorithm that derives test models that include critical transitions out of the system model for each single identified hazard in order to verify the implementation of a corresponding safety function.

Chen discussed an approach for risk-based regression testing optimization [20]. In this approach, the author applies a risk value to each test case to prioritize them. Based on these risk values, the test cases are comparable and can be prioritized to either be included in, or excluded from, a re-running regression testing process. Felderer et al. [28] describe methodology for risk-based testing that prioritizes tests on basis of a set of metrics that concisely determine testing and product related risks. Zech [29] describes a methodology for risk-based security testing in cloud computing environments. His approach uses dedicated and formalized test models to identify risks and specify negative requirements by means of so called misuse cases.

Security testing and thus, risk-based security testing could additionally gain from reusing existing test knowledge. Security test patterns are a way to formulate a solution for recurring security testing problems in a structured way where the solution of such a pattern is used in a different way each time [23]. Several patterns in the context of security testing were already defined by Smith [23] and Vouffo [24]. These patterns describe the problem or goal to be solved as well as the solution to solve this problem or to achieve this goal. They also refer to known applications, other test patterns and categorize the kind of pattern along the security approach (e.g. 'prevention' for patterns that impede unwanted incidents). However, all existing patterns have in common that they do not provide the information in a way that allow (semi-)automatic test case generation for the purpose of risk-based security testing.

With RBST and TBRA, there are at least two different ways how risk assessment and security testing can interact, but of course it should be possible to combine both, too. Erdogan et al. use a combination of both approaches [11], but it does not propose any technique or detailed guideline for how to update the risk model based on the test results.

In this paper, we describe an approach that will combine RBST, TBRA and the use of security test patterns. The approach will be presented together with a methodology specifying how it should be done in the context of a model-based testing process.

4 Combination of RA and Security Testing

TBSR and RBST can benefit from one another. Indeed, it might be helpful to switch multiple times between security testing and security risk analysis because after each round of testing and transferring the test results back into the risk picture, the risk analysis might be more precise and thus allow a better identification and prioritization of the next most critical and relevant test cases. Such an iterative process is not linear, it is an incremental process that can be visualized as a cycle. Fig. 1 illustrates our combined TBSR and RBST process.

Note that in our approach, security risk analysis is seen as both the starting point and the end point for the combined TBSR and RBST process. The main reason for this design decision is that risk analysis might also include aspects that cannot be tested while all test results can be regarded as risk analysis artifacts, too.

Security risk analysis can be conducted with any established method. In our implementation, we use the CORAS method, which is at the beginning only performed till CORAS step 6, i.e. risk estimation with threat diagrams. The results of this initial analysis are typically expressed with risk graphs or tables containing likelihood and consequence values. This initial analysis is based on literature, vulnerability databases and the system model. Its results are highly dependent on the experience and the skills of the risk analysis team. Important aspects might have been missed completely and the just guessed likelihood values are eventually very uncertain.

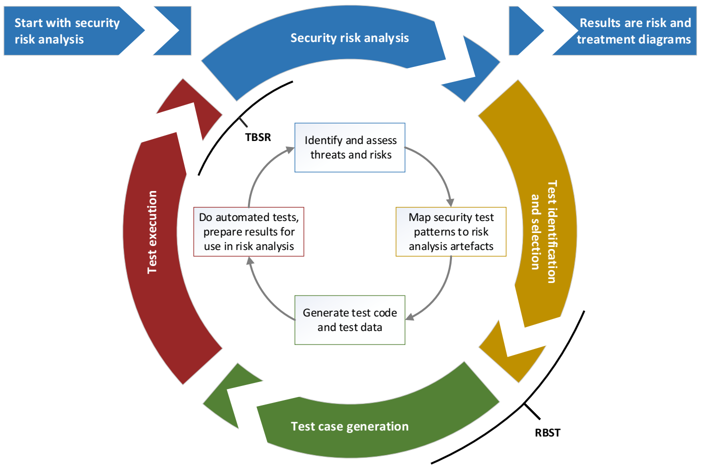

Start with security risk analysis
Security risk analysis
Results are risk and treatment diagrams
TBSR
Identify and assess threats and risks
Map security test patterns to risk analysis artefacts
Generate test code and test data
Do automated tests, prepare results for use in risk analysis
Test identification and selection
Test case generation
Test execution
RBST

Fig. 1. Combined TBSR and RBST process

4.1 Selecting Elements to Test

Though the initial analysis might be imprecise and incomplete, it is a good starting point for the first round of the security testing process, because it gives at least an idea of some things that could go wrong.

While a risk graph like the CORAS threat diagram which contains faults or unwanted incidents can be immediately interpreted as an indicator for what should be tested (i.e. trying to trigger the faults/unwanted incidents), it is not obvious how the testing should be done and which tests are the most significant. Since security testing can be expensive and since often both time and resources available for testing are rather limited, it would be most helpful to identify the most relevant test cases and to test these in the first place, at best in an automated way.

Multiple methods can be used to identify the most critical aspects that should be tested in the first place. One risk based prioritization method tries to evaluate the criticality of individual risk analysis artifacts. For example, it can use likelihood values and relations between different elements in a risk graph to calculate likelihood values for dependent incidents or faults. By setting these likelihood values in relation to the potential consequences, for any risk analysis artifact a risk value can be calculated. This approach is appropriate if the highest risks should be tested first.

Another prioritization method is motivated by the fact that risk estimates are often not precisely known. It tries to identify the impact that errors in the estimates for likelihood or consequence values for individual risk analysis artifacts would have for the entire risk picture. Besides assessments by the analysts how much confidence they have in their results and how precise these are, it is also possible to do simulations with the minimal and maximal possible values for each single risk artifact and to compare the resulting overall risk pictures, for example. This method is appropriate to focus on those elements for which the uncertainty of the risk estimation is high and the consequences of errors in the estimates are most significant.

In our combined TBSR and RBST process, only a single risk analysis artifact which should be tested next has to be selected by one of these prioritization methods.

Knowing what should be tested next is fine. However, it can be challenging to create effective test cases and to create appropriate metrics that allow sound conclusions for the risk picture. Instead of reinventing the wheel each and every time, it makes sense to create and to use a catalogue of test patterns. Ideally, the elements of a test pattern library do already contain information how they are associated with certain risk analysis artifacts.

4.2 Applying Test Patterns Using Models

Once all relevant information for the test design process are brought together, consolidated and ready for being exploited by the tester, it needs to be decided how the test design process should be carried out. In our methodology, we rely on a model-based and risk-driven approach for deriving test cases, test data, and/or test code in an automated, yet comprehensible way. In order to apply model-based techniques, a model has to be created or obtained. A model that was designed for the derivation of test artifacts is called *test model.* A test model is a "… model that specifies various testing aspects, such as test objectives, test plans, test architecture, test cases, test data etc." [21]

Models (and so are test models as well) in general are designed for a specific objective. In our methodology, the security test patterns that apply to a certain threat scenario represent these objectives for the design of a test model and, in addition, they specify what test design techniques should be applied in order to derive test cases. A conceptual model of the dependencies is depicted in Fig. 2.

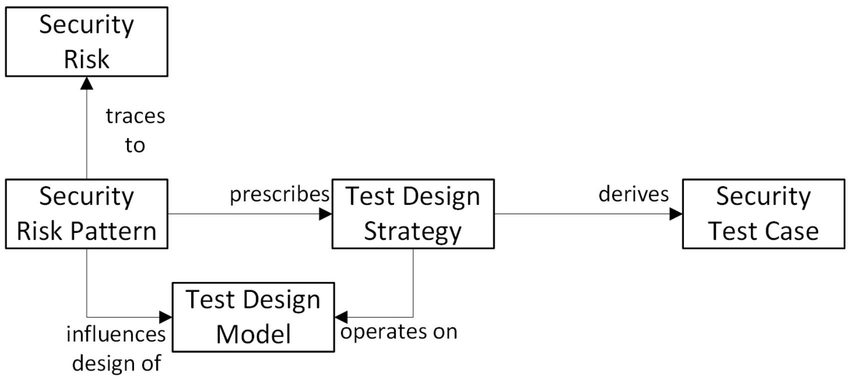


Fig. 2. Testing using patterns

Our approach to model-based test design is based on the UML Testing Profile and an additional extension for describing test design directives and test design techniques. A *test design strategy* describes a single technique to derive test cases or test data either in an automated manner (i.e., by using a test generator) or manually (i.e., performed by a test designer). A test design strategy represents the logic of a certain test design technique (such as structural coverage criteria or equivalence partitioning) and is understood as logical instructions for the entity that finally carries out the test derivation process. Examples for test design techniques standardized by ISO 29119 [27] are state transitions testing, scenario testing or the data-specific techniques boundary values analysis or equivalence partitioning. Further well-known test design techniques that are frequently applied in model-based testing are described by Utting [22].

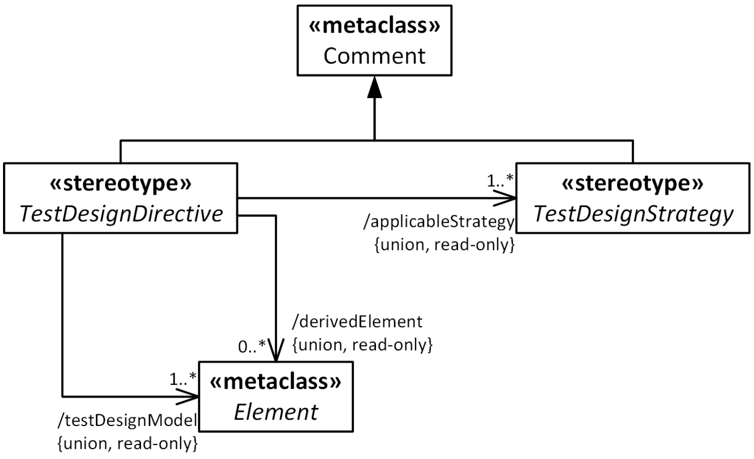


Fig. 3. UML Profile for Test Design Strategies

A *test design directive* governs an arbitrary number of test design strategies that a test generator has to obey. Therefore, it assembles appropriately deemed test design strategies to eventual fulfill the objective of a security test pattern. In risk-based security testing, we want to ensure that threat scenarios are tested thoroughly because they are deemed critical to the success of the system. In our risk-driven test design methodology, we capture the information on how to derive test cases for a specific risk (transitively referred to through security test pattern) in a test design directive in order to enable an automated derivation of test cases and test data. Directives make the entire test design activities more systematic, understandable and, even more important, reproducible. The test generation process can be easily adjusted to changed needs by just re-defining a test directive's strategy. This means that whenever the risk assessment for a threat scenario is updated, it is easily possible to adjust the intensity of the associated test design strategies and for the respective security test pattern.

The security test patterns so far are defined using natural language. This impedes (semi-) automatic test design. In order to enable test case generation from security test patterns in a (semi-) automated manner, we adapted the structure of security test pattern.

Table 1 shows an excerpt of a security test pattern. The relevant fields for a mapping between risk analysis artifacts and patterns and for test generation are marked gray. An identifier from the Common Attack Pattern Enumeration and Classification [265] allows to map identified threat scenarios from the risk model to security test patterns. In order to support different abstraction levels of risk models where identified threat scenarios are less specific, security test patterns are forming a hierarchy of generalizations. This hierarchy of patterns is realized with the field 'Generalization of'. As a consequence of more general patterns, the test strategies would be more general resulting in a larger number of generated test cases based on such a pattern.

The revised solution of a security test patterns contains, beside the solution description in natural language, two fields for (semi-)automatic test case generation: test design technique, that identifies a particular method for test case generation, and test strategies, that specifies in which way the test design technique shall be applied in order to generate test cases that are able to find the weakness the security test pattern is intended for. Such test strategies can be implemented by test case generators.

Additionally, the effort of testing for such a vulnerability as well as the effectiveness are recorded for a solution that estimate the effort for testing using the specified solution, i.e. the manual effort, and the effectiveness, i.e. how likely it is to find a vulnerability using the described solution. This information allows to select the best pattern if different patterns are applicable and resources are limited, or to prioritize several patterns. The field 'Metrics' specifies security testing metrics that allow to aggregate test results and estimate the exploitability of a revealed vulnerability based on the test results.

Table 1. Excerpt of Adapted Security Test Pattern

Pattern Name	SQL Injection	
CAPEC-ID(s)	66	
Weakness Description	*Discussion of the weakness in natural language*	
Solution	*Solution in natural language for manual testing*	
	Test Design Technique	Data fuzzing
	Test Strategies	SQL Injection
	Effort	Low to medium: can be highly automated using fuzzing techniques or SQL injection dictionaries.
	Effectiveness	Medium to high, depending on detection capabilities by access to the affected database and to error messages
Metrics	*subject of future research*	
Generalization of	Improper Input Validation	

In order to instantiate such a test pattern, several pieces of information are required. These are at least the interfaces, methods and parameters to be used by a pattern in order to stimulate the system under test. This information can be retrieved from the system under test or from a model of the system under test.

4.3 Test Result Aggregation and Integration

The final step in our combined process (i.e. the actual TBSR step) is the test result aggregation and the integration of the test results into the risk picture. Therefore, the risk analysis process restarts with the test results as new additional input information for the risk analysts. These test results might bring vulnerabilities, threat scenarios and faults / unwanted incidents to attention that were not recognized before.

Additionally, the estimation of likelihoods might become more precise taking the test results into consideration. The risk picture can be iteratively improved by starting again with selecting the next risk analysis artifact that should be studied in detail with the help of security testing.

5 Compositional Risk Analysis and Security Testing Tool

Based on our ideas for RBST in combination with TBSR we have developed a tool providing assistance for the entire process. In order to reduce the amount of manual work as far as possible, the tool tries to maximize the reusability of risk analysis artifacts and testing artifacts and to use automation where it is possible.

For risk analysis, the tool uses an extended version of CORAS supporting compositionality with the help of reusable *threat interfaces* as described in [14]. The risk graph that is generated with our tool contains besides the risk related information some information about the system that is analyzed itself, i.e. the instances of threat interfaces describe the components in detail. This information is valuable especially for automated testing. However, first of all, this system information needs to be introduced into the risk graph. Our tool automatically generates partial *threat interfaces* for components from existing compiled binaries or from source code for the components. Hence, there is no need to manually create models describing the interfaces if none are available.

The risk analysts complete the partial *threat interfaces* by adding unwanted incidents, threat scenarios and vulnerabilities. While this involves some manual work, the analysts can take advantage of our tool's assistants using existing risk related databases like CWE [26] or CAPEC [27]. Inserted risk analysis artifacts can be associated with the system model by drag and drop. For example, CWE based vulnerabilities can be dragged to input ports of threat interface instances, which automatically associates the risk graph element with the system port. The analyst is further supported with suggestions for other nodes like threat scenarios which might typically also be relevant in conjunction with already inserted nodes. For such suggested elements, even the relations to present nodes are created automatically as soon as they are inserted.

Using these assistants, negative risk analysis becomes a feasible option for analysts. In negative risk analysis, instead of trying to identify the relevant risks, initially it is assumed that all known risk artifacts (e.g. any CWE and CAPEC derived elements) are relevant. Only those that are for sure not relevant are removed. All remaining risks are considered to be relevant risks until proven otherwise.

Once the *threat interfaces* are complete with all the risks and relations that the analysts can identify without testing or simulation, the next step to verify and improve the risk picture is test identification and selection. Our tool automatically identifies appropriate test patterns for many CAPEC based threat scenarios from its test pattern library. Based on the potential consequences described in the test pattern, it is possible to generate unwanted incidents representing these consequences. By dragging the unwanted incidents to output ports of *threat interface* instances it is possible to map the unwanted incidents to components of the system that is analyzed.

In order to select the most critical scenarios to be tested in the first place, our tool can calculate likelihood values for dependent incidents using Monte Carlo Simulation. Initially, at least CWE derived vulnerabilities contain some initial default likelihood value that can be used for such calculations.

The next steps in our process are test case generation and test execution. Actually, the so far modelled graph with attached test patterns might already contain sufficient information to create and execute test cases automatically. If additional information is required, e.g. to clarify the mapping to the input ports of system under test if different mappings are possible, then our tool will ask the user to make some manual input.

How exactly the test generation and execution work will be shown in more detail below. The next step in our process is to update the risk graph with information obtained from the security testing. Metrics are required to calculate likelihood values based on the occurrence of the associated unwanted incidents observed in the tested system. The metrics should contain functions or tables setting test results and test coverage in relation to probability values per usage value (e.g. analyzed time span). Ideally, test patterns contain sound metrics. Given such metrics, our tool calculates likelihood values based on the tests and updates the risk graph with these values if the user decides to do so.

In addition to the associated unwanted incidents that have already been identified in the risk analysis and that are explicitly monitored in the security testing process, unexpected things might happen as a result of security testing, too. For example, an unexpected kind of exception could be thrown by the system under test. If our tool monitors unexpected behavior, the tool generates new unwanted incidents expressing the previously not expected behavior, which can then be inserted in the risk graph by drag and drop. The relation from the threat scenario that was tested is automatically added.

5.1 Example: Identifying and Testing Risk of Integer Overflows

In order to explore the potential as well as the limitations of our tool and to improve it further, we have created multiple sample program libraries and applications just to analyze and test them.

By dragging the *threat interface* icon to the risk graph drawing area, our tool allows to load system information from compiled programs, libraries or source code. Currently, .Net binaries and sources are fully supported. In the future, it will also analyze components from COM libraries and Java sources. For demonstration, we choose a C# written library. Our tool uses reflection to get information about exported types, functions and to generate partial *threat interfaces* for the elements the user chooses.

In our example, we choose to analyze a static function from our sample library called PrintNextNumberToString. For its one and only input parameter of type 'signed 32 bit integer', in the menu of that input port control, our tool automatically suggests a vulnerability "Integer Overflow or Wraparound", which was generated from CWE-190. The vulnerability contains an initial likelihood value "Medium" because CWE-190 says this is the likelihood that such a weakness is exploited. This likelihood information can be used for identifying the priority of testing related threat scenarios.

The menu of the CWE based vulnerability "Integer Overflow or Wraparound" contains suggestions for potentially related threat scenarios that correspond to CAPEC attack patterns. In the example, the threat scenario "Forced Integer Overflow" is suggested, which is based on CAPEC-92. The analyst can insert the threat scenario by dragging it to the risk graph. The relation from the vulnerability "Integer Overflow or Wraparound" to the "Forced Integer Overflow" threat scenario is automatically added.

The menu of the threat scenario contains a list of all applicable test patterns from our test pattern library. Each test pattern contains a list of unwanted incidents that might be the result of executing an instance of the test pattern. For test patterns related to "Forced Integer Overflow", there is an unwanted incident called "Unhandled arithmetic overflow". The unwanted incident can be dragged to the risk graph. Typically, it will be added to some output port of a function in a *threat interface* instance where the unwanted incident could be detected.

Currently, there are two test patterns available in the library of our tool to test for forced integer overflows. They differ in their strategies, directives and metrics. One test pattern only generates the extreme and special integer values like maximum, minimum and zero. The second test pattern additionally uses a data fuzzing strategy and creates a certain number of random test values. The generator of the second test pattern has multiple optional parameters. One can be used to directly set the number of fuzz test cases that should be generated. Our tool allows setting this parameter manually. However, there are also parameters available that forward values from the risk analysis and then the test pattern calculates the number of test cases that should be generated based on these values. These parameters are namely values for estimated likelihood, uncertainty, impact on the entire risk picture and the potential consequences. The higher these values are, the more test cases are generated. All values are optional. Hence, it is for example no problem if no consequence values have been estimated so far.

Both test patterns use the same idea to test for integer overflows: First, two different versions of the component that should be tested are generated. One test version that will throw an exception on any arithmetic integer overflow unless the code explicitly prevents it and one unmodified release version. If the source code is available, this is easy for any .Net program: Arithmetic overflow exceptions can be activated by a compiler switch. The tool generates the test version without requiring manual actions. Note: Though not yet implemented, it would principally also be possible to generate a test version from an IL assembly, so it could be done without access to source code, too.

Each test value is first tried with the test version of the component that will throw arithmetic overflow exceptions by default. If an overflow exception is thrown, then the same test value is tested against the release version that does not throw arithmetic overflow exceptions by default. If again the overflow exception is observed, then the release version detects the overflow correctly. Of course, when the tested function is called, the overflow exceptions must be treated properly, but throwing the exception itself in the release version is not considered to be an error, it is not necessarily an unwanted incident. If treated correctly by the caller, the program might continue without problems.

In contrast, if only the *test version* throws an overflow exception and the *release version* does not throw an overflow exception, then the *release version* calculates eventually wrong values and there is probably no way to detect the error.

Besides detecting arithmetic overflow exceptions, any other exception that occurs during the test is regarded to be an unexpected exception and it is reported as a potentially new unwanted incident to our tool. Fig. 4 shows how the testing is evaluated.

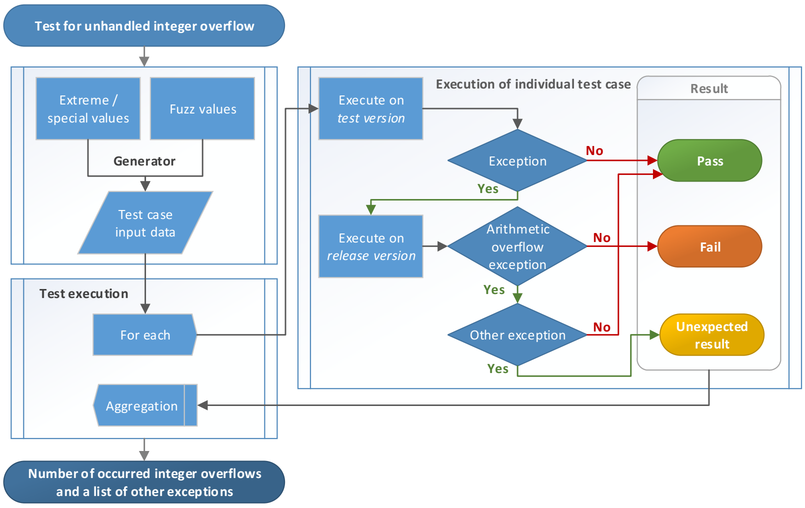

Test for unhandled integer overflow
Extreme / special values
Fuzz values
Generator
Test case input data
Test execution
For each
Aggregation
Number of occurred integer overflows and a list of other exceptions
Execution of individual test case
Execute on test version
Exception
No
Yes
Execute on release version
Arithmetic overflow exception
No
Yes
Other exception
No
Yes
Result
Pass
Fail
Unexpected result

Fig. 4. Testing for Integer overflows

Our tool compiles code taken or automatically generated from test patterns. For actually executing the test cases, the risk graph is evaluated to identify which functions have to be called with which parameters and what has to be monitored. In the example we present here, this requires no additional manual work at all.

In our sample library, there are multiple functions that can be tested for forced integer overflows. Some are more complex and do expect more than just a single input value. To test such functions, it is necessary to assign test case generators to each input parameter. This can be done by assigning threat scenarios and by choosing test patterns. Then, only one of the test patterns is actually tested at a time, i.e. its test strategies are compiled and executed. From the other test patterns, only the test data generators are used. Since by default all possible combinations of the generated test values for each parameter are tested, the amount of test cases can grow very quick by doing so. Alternatively, our tool also allows to create data generators that just produce test data. However, these do require manual writing of code. Finally, there are constant values assignable in an easy way.

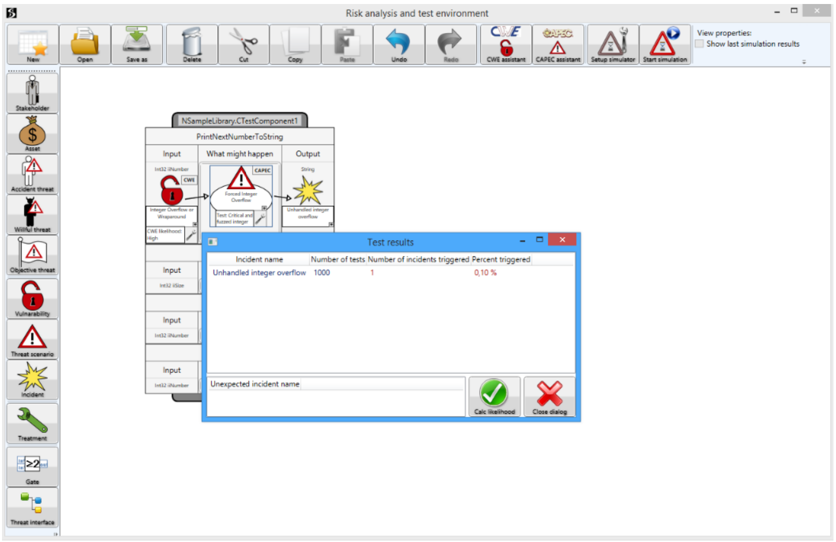
Both test patterns from our library that are applicable to test for integer overflows contain a metric which can be used to calculate the likelihood that an attacker will produce an integer overflow by calling the tested function. For the test pattern that uses fuzzing, the metric uses the total number of test cases that are generated and executed to calculate coverage. The risk graph can be updated with the new likelihood value if the analyst confirms it.

5.2 First Evaluation

In the optimal case, our tool assists the risk analysts and testers so much that only a few mouse actions are required to do combined risk analysis with automated security tests. The example shows that no single line of code has to be written, no test cases have to be created manually and no manual interpretation of the test results is required if well-defined test patterns are available and applied correctly.

Using our tool, modeling of the risk graph remains a manual task. But the analyst is great much assisted with existing artifacts from libraries and artifacts generated based on test results. Likelihoods can automatically be calculated based on test results. Furthermore, for dependent incidents, likelihood values can be calculated.

Though our tool is in an early phase of development, it already provides an intuitive workflow. Fig. 5 shows the UI of our tool. It proves to save some amount of work and to reduce dependency upon the skills and the judgment of the analysts and testers.



Incident name	Number of tests	Number of incidents triggered	Percent triggered
Unhandled integer overflow	1000	1	0,10 %

Unexpected incident name

Fig. 5. Screenshot of the tool analyzing and testing the example

Currently, our test pattern library is pretty small and it is quite a challenge to develop test patterns that are general and flexible enough to be applicable for diverse systems and that can though be instantiated without lots of manual work – especially without manually writing code. Hence, practical usability is for now limited to relatively few cases.

The tool we developed is a standalone application. However, its core is an API which can be integrated in and used by other tools. Indeed, it will be possible to use our tool only for parts of the combined risk analysis and security testing process.

6 Conclusion, Ongoing and Future Work

Though there are lots of technologies and tools for risk assessment and security testing, applying them for large complex systems is still a challenge.

Development of the methodology and the tool described here is still in an early stage. Our efforts are driven by case studies. These provide use cases and requirements inspiring our development. We plan to test and to evaluate our method and our tool by using them within these case studies. Additionally, we will analyze the same use cases with other existing methods and tools so that we can compare the results in relation to the effort for the different approaches.

Our vision is that risk assessment should become a process that typically takes place in an open collaboration. Risk analysis results and test patterns should be made accessible for anybody as reusable artifacts. We plan to create a public open database for that purpose. This data would be helpful for other developers reusing the analyzed component as they could integrate the risk analysis artifacts in their own compositional risk assessment for their products. Reusing the test patterns could reduce testing costs and improve testing quality. The end users could benefit from such a database, too, because they could inform themselves about the remaining risks in a standardized way.

References

1. International Organization for Standardization: ISO 31000 Risk management – Principles and guidelines (2009)
2. International Organization for Standardization: ISO Guide 73 Risk management – Vocabulary (2009)
3. Bouti, A., Kadi, D.A.: A state-of-the-art review of FMEA/FMECA. International Journal of Reliability, Quality and Safety Engineering 1, 515–543 (1994)
4. International Electrotechnical Commission: IEC 61025 Fault Tree Analysis (FTA) (1990)
5. International Electrotechnical Commission: IEC 60300-3-9 Dependability management – Part 3: Application guide – Section 9: Risk analysis of technological systems – Event Tree Analysis (ETA) (1995)
6. Lund, M.S., Solhaug, B., Stølen, K.: Model-Driven Risk Analysis – The CORAS Approach. Springer (2011)
7. Lund, M.S., Solhaug, B., Stølen, K.: Evolution in relation to risk and trust management. Computer 43(5), 49–55 (2010)
8. Kaiser, B., Liggesmeyer, P., Mäckel, O.: A new component concept for fault trees. In: 8th Australian Workshop on Safety Critical Systems and Software (SCS 2003), pp. 37–46. Australian Computer Society (2003)
9. Papadoupoulos, Y., McDermid, J., Sasse, R., Heiner, G.: Analysis and synthesis of the behaviour of complex programmable electronic systems in conditions of failure. Reliability Engineering and System Safety 71(3), 229–247 (2001)
10. Erdogan, G., Li, Y., Runde, R.K., Seehusen, F., Stølen, K.: Conceptual Framework for the DIAMONDS Project. Oslo (May 2012)

11. Erdogan, G., Seehusen, F., Stølen, K., Aagedal, J.: Assessing the usefulness of testing for validating the correctness of security risk models based on an industrial case study. In: Proc. Workshop on Quantitative Aspects in Security Assurance (QASA 2012), Pisa (2012)
12. Benet, A.F.: A risk driven approach to testing medical device software. In: Advances in Systems Safety, pp. 157–168. Springer (2011)
13. Kloos, J., Hussain, T., Eschbach, R.: Risk-based testing of safety-critical embedded systems driven by fault tree analysis. In: Software Testing, Verification and Validation Workshops (ICSTW 2011), pp. 26–33. IEEE (2011)
14. Viehmann, J.: Reusing Risk Analysis Results - An Extension for the CORAS Risk Analysis Method. In: 4th IEEE International Conference on Information Privacy, Security, Risk and Trust (PASSAT 2012), Amsterdam, pp. 742–751. IEEE (2012)
15. Bach, G.J.: Heuristic Risk-Based Testing. Software Testing and Quality Engineering Magazine, 96–98 (November 1999)
16. Stallbaum, H., Metzger, A.: Employing Requirements Metrics for Automating Early Risk Assessment. In: Proceedings of the Workshop on Measuring Requirements for Project and Product Success, MeReP 2007, at Intl. Conference on Software Process and Product Measurement, Spain, pp. 1–12 (2007)
17. Stallbaum, H., Metzger, A., Pohl, K.: An Automated Technique for Risk-based Test Case Generation and Prioritization. In: Proceedings of 3rd Workshop on Automation of Software Test, AST 2008, Germany, pp. 67–70 (2008)
18. Bauer, T., et al.: From Requirements to Statistical Testing of Embedded Systems. In: Software Engineering for Automotive Systems (ICSE), pp. 3–10 (2007)
19. Zimmermann, F., Eschbach, R., Kloos, J., Bauer, T.: Risk-based Statistical Testing: A Refinement-based Approach to the Reliability Analysis of Safety-Critical Systems. In: Proceedings of the 12th European Workshop on Dependable Computing (EWDC), France (2009)
20. Chen, Y., Probert, R., Sims, P.: Specification-based Regression Test Selection with Risk Analysis. In: Proceedings of the 2002 conference of the Centre for Advanced Studies on Collaborative research (CASCON 2002), p. 1 (2002)
21. Object Management Group (OMG): UML Testing Profile, http://www.omg.org/spec/UTP
22. Utting, M., Legeard, B.: Practical Model-based testing – A Tools Approach. Elsevier (2007)
23. Smith, B.: Security Test Patterns (2008), http://www.securitytestpatterns.org/doku.php
24. Vouffo Feudjio, A.-G.: Initial security test patterns catalogue. DIAMONDS project deliverable D3.WP4.T1
25. MITRE: Common Attack Pattern Enumeration and Classification (2014), http://capec.mitre.org
26. MITRE: Common Weakness Enumeration (2014), http://cwe.mitre.org
27. International Organization for Standardization: ISO/IEC 29119-1 Systems and software engineering—Software testing—Part 1: Concepts and definitions (2013)
28. Felderer, M., Haisjackl, C., Breu, R., Motz, J.: Integrating manual and automatic risk assessment for risk-based testing. In: Biffl, S., Winkler, D., Bergsmann, J. (eds.) SWQD 2012. LNBIP, vol. 94, pp. 159–180. Springer, Heidelberg (2012)
29. Zech, P., et al.: Towards a model based security testing approach of cloud computing environments. In: 2012 IEEE Sixth International Conference on Software Security and Reliability Companion (SERE-C). IEEE (2012)

Risk-Based Vulnerability Testing Using Security Test Patterns

Julien Botella[1], Bruno Legeard[1,2], Fabien Peureux[1,2], and Alexandre Vernotte[2]

[1] Smartesting R&D Center - 2G, Avenue des Montboucons, 25000 Besançon, France
`{botella,legeard,peureux}@smartesting.com`

[2] Institut FEMTO-ST, UMR CNRS 6174 - Route de Gray, 25030 Besançon, France
`{blegeard,fpeureux,avernott}@femto-st.fr`

Abstract. This paper introduces an original security testing approach guided by risk assessment, by means of risk coverage, to perform and automate vulnerability testing for Web applications. This approach, called Risk-Based Vulnerability Testing, adapts Model-Based Testing techniques, which are mostly used currently to address functional features. It also extends Model-Based Vulnerability Testing techniques by driving the testing process using security test patterns selected from risk assessment results. The adaptation of such techniques for Risk-Based Vulnerability Testing defines novel features in this research domain. In this paper, we describe the principles of our approach, which is based on a mixed modeling of the System Under Test: the model used for automated test generation captures some behavioral aspects of the Web applications, but also includes vulnerability test purposes to drive the test generation process.

Keywords: Risk-Based Testing, Security test pattern, Model-Based Testing, Web application vulnerability, CORAS, SQL Injection.

1 Introduction

Based on the current state of the art on security and on all the security reports like OWASP Top Ten 2013 [1], CWE/SANS 25 [2] and WhiteHat Website Security Statistic Report 2013 [3], Web applications are the most popular targets when speaking of cyber-attacks. The fact that modern society relies on the Web a little more everyday foregrounds the challenges of IT security, particularly in terms of data privacy, data integrity and service availability.

The mosaic of technologies used in current Web applications (e.g., HTML5 and JavaScript frameworks) increases the risk of security breaches. This situation has led to significant growth in application-level vulnerabilities, with thousands of vulnerabilities detected and disclosed annually in public databases such as the MITRE CVE - Common Vulnerabilities and Exposures [2]. The most common vulnerabilities found on these databases especially emphasize the lack of resistance to code injection of the kind SQL Injection (SQLI) or Cross-Site Scripting (XSS), which have many variants. This kind of vulnerabilities indeed appears in the top list of current Web applications attacks.

Application-level vulnerability testing is first performed by developers, but they often lack the sufficient in-depth knowledge in recent vulnerabilities and related exploits. This kind of tests can also be achieved by companies specialized in security testing, in penetration testing for instance. These companies monitor the constant discovery of such vulnerabilities, as well as the constant evolution of attack techniques. But they mainly use manual approaches, making the dissemination of their techniques very difficult, and the impact of this knowledge very low. Finally, Web application vulnerability scanners can be used to automate the detection of vulnerabilities, but since they often generate many false positive and false negative results, human investigation is also required [4,5].

This paper proposes a Risk-Based Vulnerability Testing (RBVT) approach in order to improve the overall level of Web application security by increasing the accuracy and precision of security testing according to the risk assessment. To achieve this goal, the approach consists to drive the test generation strategy using risk metrics and relevant vulnerability test patterns, which are directly related to risk assessment process of the System Under Test (SUT). This objective also includes the development of a tool supporting this RBVT process in order to automate the detection of such vulnerabilities, and therefore to get feedback about risk assessment. Hence the main contributions of the paper relate to the proposal of a risk-based and pattern-driven approach to generate vulnerability test cases for Web applications. More precisely, they are the following:

– Techniques addressing both risk-based test identification, by means of vulnerability test patterns, and test prioritization to drive the overall testing generation process.
– The extension of a test purpose language to drive the test generation engine through models in order to cover the targeted vulnerability test patterns.
– The full automation, ensuring risk traceability, of the test purpose selection (given by the risk model), test case execution and verdict assignment from risk assessment results.

The paper is organized as follows: Section 2 introduces the context and the principles of the RBVT approach. Section 3 details the use of the RBVT by describing the content of the testing input artefacts, the test pattern language and the risk-driven test generation, which is illustrated using a simple example of Web application, namely eCinema. Related work about vulnerability detection is discussed in Section 4. Finally, conclusion and future works are given in Section 5.

2 Context and Principles of the RBVT Approach

Model-Based Testing (MBT) [6] is a software testing approach in which both test cases and expected results are automatically derived from an abstract model of the SUT. MBT is usually performed to automate and rationalize functional black-box testing. It is a widely-used approach that has gained much interest in recent years, from academic as well as industrial domains, especially by increasing and mastering test coverage, including support for certification, and by providing the degree of automation needed for accelerating the test process [7].

More precisely, MBT techniques derive abstract test cases (including stimuli and expected outputs) from an abstract model, and enable the generation of executable tests from these abstract test cases. The abstract model, called test model, formalizes the behavioural aspects of the SUT in the context of its environment and at a given level of abstraction. It thus captures the control and observation points, the expected dynamic behaviour, the data associated with the tests, and finally the initial state of the SUT. The test cases generated from such models allow to validate the functional aspects of the SUT by comparing back-to-back the results observed on the SUT with those specified by the model. Therefore MBT aims to ensure that the final product conforms to the initial functional requirements. However, if these techniques are used to cover the functional requirements specified in the test model of the SUT, they are also limited to this scope since what is not modeled cannot be tested.

The proposed approach to perform vulnerability testing is based on MBT process, and is thus composed of the four activities depicted in Figure 1:

① the *Test Purposes* activity consists of formalizing test purposes from vulnerability test goals that the generated test cases have to cover;
② the *Modeling* activity aims to define a model that captures the behavioral aspects of the SUT in order to generate consistent (from a functional point of view) sequences of stimuli;
③ the *Test Generation* activity comprises the automated production of abstract test cases from the artefacts defined during the two previous activities;
④ the *Adaptation, Test Execution and Observation* activity aims (i) to translate the generated abstract test cases into executable scripts, (ii) to execute these scripts on the SUT, (iii) to observe the SUT responses and to compare them to the expected results in order to assign the test verdict and automate the detection of vulnerabilities.

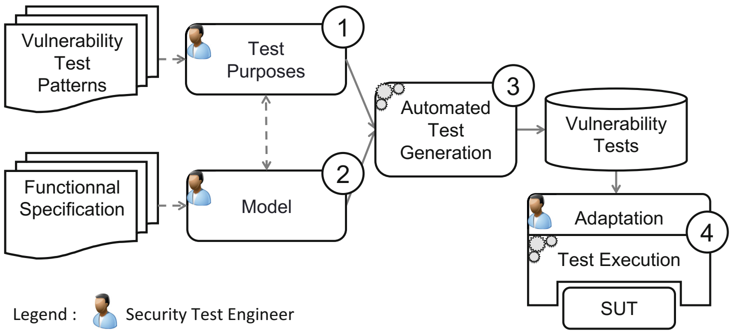

Vulnerability Test Patterns
Test Purposes
1
Automated Test Generation
3
Vulnerability Tests
Functionnal Specification
Model
2
Adaptation
4
Test Execution
SUT
Legend : Security Test Engineer

Fig. 1. Model-based vulnerability testing process

For a further description of each activities of this model-based vulnerability testing process, a detailed presentation can be found in [8].

All these activities are supported by a dedicated toolchain, which is based on an existing MBT software named *CertifyIt* [9] provided by the company Smartesting[1]. This software is a test generator that takes as input a test model, written with a subset of UML/OCL (called UML4MBT [10]), which captures the behavior of the SUT. Concretely, a UML4MBT test model consists of (i) UML class diagrams to represent the static view of the system (with classes, associations, enumerations, class attributes and operations), (ii) UML Object diagrams to list the concrete objects used to compute test cases and to define the initial state of the SUT, and (iii) state diagrams (annotated with OCL constraints) to specify the dynamic view of the SUT. OCL expressions provide the expected level of formalization necessary for model-based testing modeling since an operational interpretation of OCL postconditions makes it possible to determine its effect (this specific interpretation of OCL, called OCL4MBT [10], basically consists to interpret OCL equality as an assignment). That is why such UML4MBT test models have a precise and unambiguous meaning, so that these models can be understood and processed by the *CertifyIt* technology. This precise meaning makes it possible to simulate the execution of the test models and to automatically generate test cases by applying predefined coverage strategies or by applying test directives formalized by a dedicated test purpose language.

A *test purpose* is a high-level expression that formalizes a test intention linked to a test objective to drive the automated test generation on the test model. This is a textual language based on regular expressions, allowing the formalization of vulnerability test intention in terms of states to be reached and operations to be called. This test purpose language has been originally designed to drive model-based test generation for security components, typically Smart card applications and cryptographic components [11]. This test purpose language has been extended to be able to formalize typical vulnerability test patterns for Web applications in conjunction with generic and specific test models.

Each of such generated test cases is typically an abstract sequence of high-level actions (operations) specified in the UML4MBT test models. These generated test sequences contain the sequence of stimuli to be executed, but also the expected results (to perform the observation activity), obtained by resolving the associated OCL4MBT constraints. About this vulnerability testing process, it should be noted that, within the traditional MBT process that allows to generate functional test cases, positive test cases are computed to validate the SUT in regards to its functional requirements. We call "positive test" a test case that checks whether a sequence of stimuli produces the expected effects with regards to the specifications. When a positive test is in success, it demonstrates that the tested scenario is implemented correctly. Within vulnerability testing approach, "negative test cases" have to be produced: typically, attack scenarios to obtain data from the SUT in an unauthorized manner. A negative test case thus targets an unexpected use of the SUT in order to show that the SUT allows something that it is not supposed to allow. In our approach, when a negative test case succeeds, it highlights a problem in the SUT.

[1] `http://www.smartesting.com`

We propose to drive this vulnerability testing process by risk assessment in order to perform and automate risk-based testing for Web applications. *Risk & requirements-based testing* was originally the title of an article from James Bach [12]. This article was underlining the creative aspects of software testing to manage stated and unstated requirements depending on risks associated with the SUT. Risk may be defined as the combination of the impact of the severity (consequence) and the likelihood (probability) of a hazardous failure of the SUT. A risk-based testing management method focuses on risk assessment and test prioritization based on requirements. Within MBT, this approach influences the entire testing process, and has the following impacts:

- Risk analysis drives the development and maintenance of test generation artefacts: the level of detail as well as the scope of test generation models are determined according to established priorities. This impacts the test models, which have to capture risk aspects besides functional features.
- During the test generation phase, test selection criteria applied on the test models are specified to cover risk and priorities for requirements coverage.

Therefore, MBT allow to implement risk-based testing in the modeling phase by adapting modeling effort to risk analysis and assessment, and in the test generation phase by adapting test selection criteria to risk-based test priorities. RBVT aims to integrate the existing model-based vulnerability testing approach with risk-based testing approach. Concretely, it consists somehow to drive the test generation regarding risk assessment results and using dedicated vulnerability test patterns. The RBVT overall testing process is depicted in Figure 2.

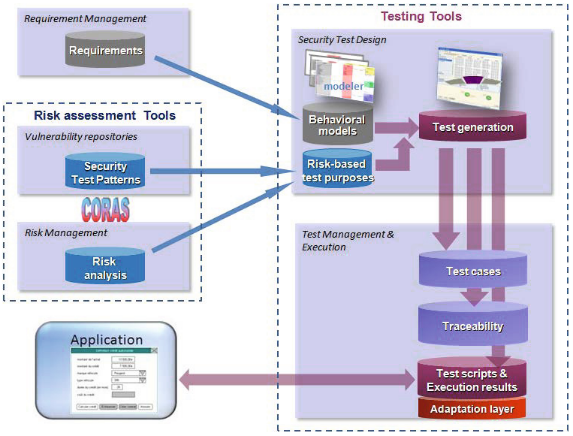


Fig. 2. Risk-Based Vulnerability Testing overall process

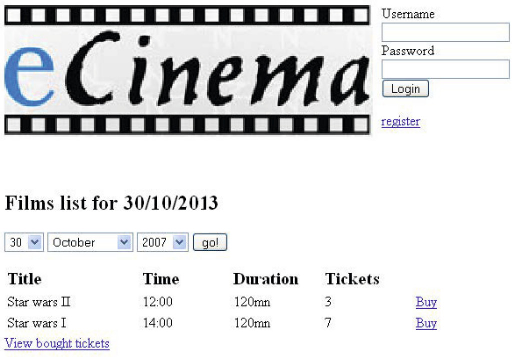
RBVT process starts by risk analysis, for example with an approach such as CORAS [13], which provides a customized language for threat and risk modeling. Security test patterns based on identified and prioritized vulnerabilities from the risk analysis provide a starting point for test case generation: they indeed link the risk analysis results and security testing goal by providing information how relevant vulnerability test cases can be derived from risk assessment.

In order to generate these expected vulnerability test cases, test cases are automatically derived from a formalization of the security test patterns using the test purpose language. Finally, the last step consists to export the abstract test cases into an execution environment, in which they are concretized using a dedicated adaptation layer to be executed. Moreover, this process makes it possible to manage the traceability between the targeted security test patterns (formalized with test purposes) and the associated generated test cases. This management is performed through the automated generation, during the test generation process, of a traceability matrix that links vulnerabilities to generated test cases. To support this RBVT process, the *CertifyIt* technology has been extended by the following developments:

– Import of the risk analysis results. It enables to select the related test purposes and to prioritize them regarding risk identification and estimation.
– Test purpose language extensions. On the one hand, the definition of keywords enables to provide generic Test Purposes related to security test patterns, and to help for maintenance and reuse. On the other hand, a mechanism to link a Test Purpose to a requirement identifier has been created to ensure the traceability through the all test generation process.
– Test Purpose catalogue import/export. It makes it possible to reuse and apply generic Test Purposes on several SUT.

3 Applying the RBVT Approach

In this section, we detail each activity of the process introduced in Figure 2, including the features introduced at the end of the previous section. For each activity, we present its objectives as well as the tooling that automates it. The eCinema running example is used to illustrate our statements. Basically, eCinema is a simple Web application that allows a customer to buy tickets on line before to go to his favorite cinema. The welcome screen, depicted in Figure 3, displays the list of available movies and show times.



Films list for 30/10/2013

Title	Time	Duration	Tickets	
Star wars II	12:00	120mn	3	Buy
Star wars I	14:00	120mn	7	Buy

Fig. 3. eCinema welcome screen

Before selecting tickets, a user should be logged to the system. This requires a registration. A registration is valid when a user gives a name (not already used) and a valid password. A valid new registration implies that the user is automatically logged in. When logged in, the user can buy tickets. If tickets are available, he can buy some of them and see his basket to verify his selection. When checking his selection, the user can delete tickets and then the number of available tickets for the session is automatically updated.

3.1 Selection and Prioritization of Vulnerabilities from Risk Analysis

The starting point of the process is the identification and prioritization of the vulnerabilities, which are defined by a risk analysis activity. These results are indeed used to drive the test generation strategy. Our approach is based on the CORAS risk assessment method [13]. CORAS is a model-driven method for risk analysis featuring a tool-supported modelling language especially designed to model risks that are common for a large number of systems. Such models serve as a basis to perform risk identification and prioritization. For example, Figure 4 shows an example of CORAS threat diagram describing SQL Injection vulnerability that can occur when a user is logging the eCinema Web application. In this context, due to the *insufficient user validation* threat, *SQL Injection successful* is a threat scenario and can lead to the unwanted incident defined by the *disclosure of confidential information*. The likelihood of the threat is considered as *possible* and its consequence *moderate*.

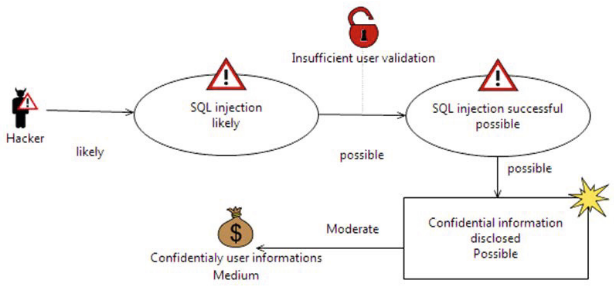


Fig. 4. CORAS model example for SQL Injection vulnerability

Each identified threat scenario is linked to a dedicated vulnerability test pattern (vTP). A vTP defines the testing procedure allowing the detection of the corresponding threat in a Web application. There are as much vTP as there are types of application-level breaches. The ITEA2 DIAMONDS[2] research project provided a first definition, as well as a first listing of vTP [14], which has been

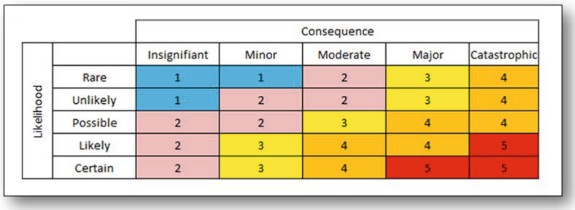
[2] `http://www.itea2-diamonds.org`

Name	SQL Injection
CWE-ID(s)	CWE-89
Description	The software constructs all or part of an SQL command using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the intended SQL command when it is sent to a downstream component.
Objective(s)	Based on attack pattern CAPEC-66 1. Use the application, client or Web browser to inject SQL constructs input through text fields or through HTTP GET parameters. 2. Use a possibly modified client application or Web application debugging tool such to submit SQL constructs for submitted values or to modify HTTP POST parameters, hidden fields, non-free form fields, etc. 3. Check for error messages, delays, disclosed values in the client application and new / modified / deleted values in the database. Detect if a user input can embed malicious datum enabling a Reflected XSS attack.
Test Data	SQL Injection Cheat Sheet
...	...
References	OWASP Top 10 (2013): A1-Injection, CAPEC-7: Blind SQL Injection, CAPEC-66: SQL Injection, OWASP Testing Guide: Testing for SQL Injection (OWASP-DV-005), OWASP: Automated Audit using SQLMap

Fig. 5. Vulnerability test pattern for SQL Injection

extended for test generation needs. Figure 5 presents an excerpt of the vulnerability test pattern defining the SQL Injection.

The vulnerability test patterns that have to be used by the test generation algorithm are gathered from the threat scenarios of each CORAS model related to the SUT. Moreover, likelihood and consequence are also collected from the CORAS model to assign a priority to the threat scenarios, and thus to prioritize them. Figure 6 shows the risk assessment matrix that enables to set such priority.



		Consequence				
		Insignifiant	Minor	Moderate	Major	Catastrophic
Likelihood	Rare	1	1	2	3	4
	Unlikely	1	2	2	3	4
	Possible	2	2	3	4	4
	Likely	2	3	4	4	5
	Certain	2	3	4	5	5

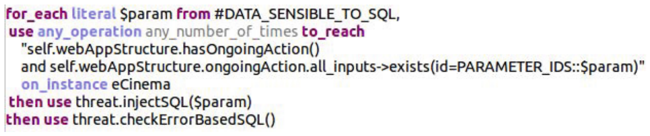
Fig. 6. Risk evaluation matrix

The assigned priority level (from 1 to 5) will be used during test case generation to select the coverage of the test purpose (priority 1 defines the lower coverage and so less generated test cases, whereas priority 5 defines the higher coverage and so more generated test cases). The CWE identifiers and the corresponding priority levels are then exported to *CertifyIt* in order to drive the test generation process, which is presented in the next subsections.

3.2 Formalizing Vulnerability Test Patterns into Test Purposes

In the test generation tool, dedicated and generic test purposes make it possible to formalize each vTP imported from risk assessment. A test purpose is a high level expression that formalizes a test intention linked to a test objective to drive the automated test generation on the test model. It allows the formalization of vulnerability test intention in terms of states to be reached and operations to be called. The language relies on combining keywords, to produce expressions that are both powerful and easy to read. Basically, a test purpose is a sequence of major *stages* to be reached. A stage is a set of operations or behaviors to use, or/and a state to reach. Transforming the sequence of stages into a complete test case, based on the test model, is left to the MBT technology (more details will be given in subsection 3.4). Furthermore, at the beginning of a test purpose, are defined *iterators* that are used in the stages in order to introduce context variations (the threat priority exported from CORAS model is then used to set a given level of variation combinations). Each combination of possible values of iterators produces a specific test case.

Figure 7 shows the instantiated test purpose formalizing the vTP of Figure 5. This schema precises that for all malicious data enabling the detection of SQL Injection and from all sensible Web pages, it is required to do the following actions: (i) use any operation to activate the sensible page, (ii) inject malicious data in all the user inputs of the page, (iii) check if the page is sensible to the attack. The keywords ALL_* define enumerations of values allowing to master the final amount of test cases regarding test priority.



```
for_each literal $param from #DATA_SENSIBLE_TO_SQL,
use any_operation any_number_of_times to_reach
  "self.webAppStructure.hasOngoingAction()
  and self.webAppStructure.ongoingAction.all_inputs->exists(id=PARAMETER_IDS::$param)"
  on_instance eCinema
then use threat.injectSQL($param)
then use threat.checkErrorBasedSQL()
```

Fig. 7. Test purpose formalizing the SQL Injection vTP (of Figure 5)

Finally, variants of malicious data are defined during the modeling activity, variants of the procedure are defined during the adaptation and execution activity. In order to generate tests from models, the test purposes is used in conjunction with the test model, which is introduced in the next subsection.

3.3 Modeling

As for every MBT approach, the modeling activity consists of designing a test model that will be used as basis to generate the abstract test cases. This model uses the UML notation to represent the Web application to be tested. We will see that some parts of the model are generic and re-usable for modeling any Web applications, while some other parts are specific to the Web application that is considered. We present in the following the used UML diagrams (classes, objects, statechart diagrams), and their respective use in the context of our approach.

Class diagrams specify the static aspect of the model, by defining in an abstract manner the structure and entities managed by the SUT. *Classes* model business objects. *Associations* model relations between business objects. *Enumerations* model sets of abstract values, and *literals* model each value. *Class attributes* model evolving characteristics of business objects. *Class operations* model points of control and observation of the SUT (we describe here the navigation between pages). In the context of Web applications, the model presents some generic parts, shown in Figure 8, which are the same for all considered Web applications:

- four classes (*WebAppStructure*, *Page*, *Action* and *Data*) and their associations respectively model the general structure of the application, the available pages (or screens in case single-URL applications), the available actions on each page, and the user inputs of each action potentially used to inject an attack vector (i.e. malicious data to perform the attack). The login page of an application is modeled using:
  - a particular *'Login' Page*, modeling the application's login page;
  - a particular *'Login' Action*, modeling the sending of the form to the server
  - two particular *Data*, modeling the user's name and password
- the *Threat* class models the potential threats: its operations *injectXSS()* and *checkXSS()* model the means to exercise and observe the attack.

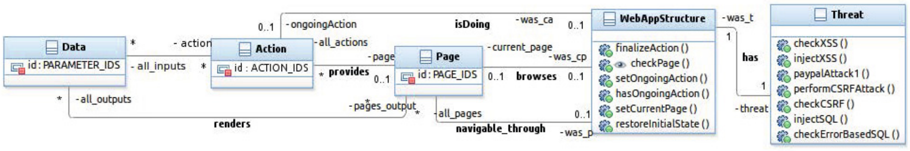


Fig. 8. Generic class diagram of the SUT structure

Figure 9 presents the class model of the eCinema example. These classes are eCinema specific classes, and are in addition to the generic classes presented in Figure 8. This class diagram displays the additional classes *ECinema* and *User* to respectively model the SUT and its potential users.

The UML statechart diagram graphically represents the behavioral aspect of the SUT, modeling the navigation between pages in the Web applications. *States* model Web pages, and *transitions* model the available links between these Web pages (HTML links, form submissions, etc.). *Triggers* of transitions are the UML operations of the SUT class. *Guards* of transitions (specified using OCL4MBT) precisely define the execution context of the transition. Finally, the *effects* of the transitions (also specified using OCL4MBT) precisely describe its expected behavior that should be modeled for vulnerability test generation. Figure 10 presents the statechart diagram of the eCinema example.

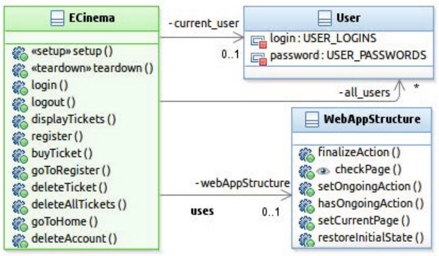


Fig. 9. eCinema-specific class diagram

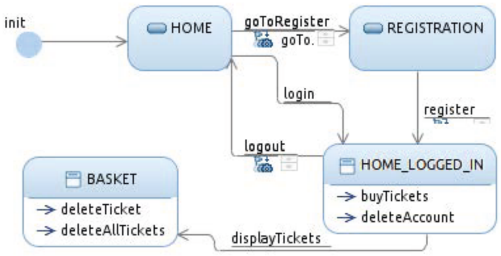


Fig. 10. eCinema statechart diagram

The UML object diagram models the initial state of the SUT by instantiating the class diagram: the *instances* model business entities available at the initial state, and the *links* instantiate the associations between these instances. In our approach, the object diagram models the Web pages and the user inputs of these pages. Figure 11 presents the initial state of the eCinema example. It specifies: (*a*) one user, with its credentials, and (*b*) the pages and user inputs of eCinema.

These last two parts (namely, the statechart diagrams and the object diagrams) are necessarily specific to each considered application.

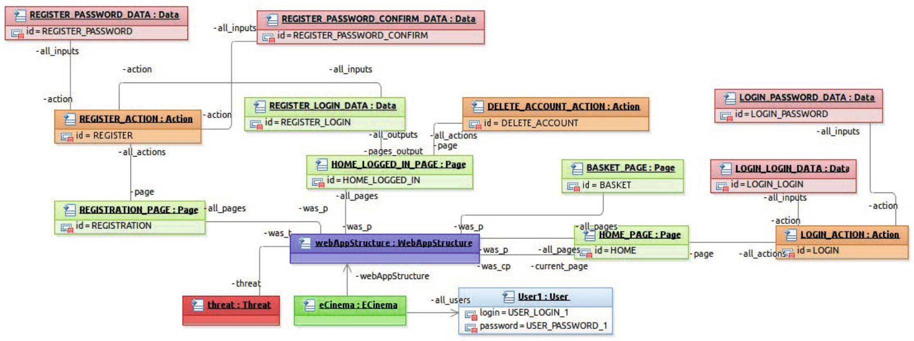


Fig. 11. eCinema object diagram for the initial state

3.4 Test Generation and Execution

The main purpose of the *test generation* activity is to produce test cases from both the model and the test purposes. Three phases compose this activity. The first phase transforms the model and the test purposes into elements computable by the *CertifyIt* MBT tool. Notably, test purposes are transformed into *test targets*, which can be seen as a sequence of *intermediate objectives* used by the symbolic generator. Hence, the sequence of stages of a test purpose is mapped to a sequence of intermediate objectives of a test target. Furthermore, this first phase manages the combination of values between iterators of test purposes, such that one test purpose produces an amount of test targets depending of the priority level calculated during risk assessment. The generator respectively calculates the first N combinations such that $N = maxCombination/(5 - priority + 1)$ where $maxCombination$ and $priority$ respectively denote the maximum amount of possible combinations and the priority level. For example, if the priority is 5 (the higher), all the combinations of values between iterators are expanded.

The second phase produces the *abstract test cases* from the test targets. This phase is left to the test case generator. An abstract test case is a sequence of *steps*, where a step corresponds to a completely valued operation call. An operation call represents either a stimulation or an observation of the SUT. Each test target produces one test case (i) verifying the sequence of intermediate objectives and (ii) verifying the model constraints. Note that an intermediate objective (and hence, a test purpose stage) can be transformed into several steps.

Finally, the third phase exports the abstract test cases into the execution environment. In our case, it consists of (i) creating a JUnit test suite, where each abstract test case is exported as a JUnit test case, and (ii) creating an interface. This interface defines the prototype of each operation of the SUT. The implementation of these operations is in charge of the test automation engineer.

In the test model, all data used by the application (page, user input field, malicious datum, user credentials, etc.) are specified in an abstract way. Hence, the test suite cannot be executed as it is. The gap between abstract keywords used in abstract test cases and the real API of the SUT must be filled. Stimuli must also be adapted. When exporting abstract test cases, the MBT tool provides an interface defining each operation signature. The test automation engineer is in charge to implement the automated execution of each operation of this interface. Since we are testing Web applications, two ways of automation are proposed:

– the *GUI* level: we stimulate and observe the application *via* the client-side GUI of the application. Even if this technique is time consuming, it could be necessary when the client-side part of the application embeds JavaScript scripts. For this technique, Selenium framework is used.
– the *HTTP* level: we stimulate and observe the application *via* HTTP messages send to (and received from) the server-side application. This technique is extremely fast and can be used to bypass HTML and JavaScript limitations. For this technique, we are using the Apache HTTPClient Java library.

4 Related Work

Related work on vulnerability detection can be classified into two categories: static and dynamic analysis security testing. Static Application Security Testing (SAST) are white-box approaches including source, byte and object code scanners and static analysis techniques. Dynamic Application Security Testing (DAST) includes black-box web application scanners, fuzzing techniques and emerging model-based security testing approaches. In practice, these techniques are complementary, addressing different types of vulnerabilities. For example, SAST techniques are known to be efficient to detect buffer overflow and badly formatted string, but weak to detect SQLI, XSS or CSRF vulnerabilities. RBVT is a dynamic testing technique, so this section focuses on DAST techniques by providing a state of the art of emerging model-based security testing techniques.

Web application vulnerability scanners aim to detect vulnerabilities by injecting attack vectors. These tools generally include three main components [15]: a crawler module to follow Web links and URLs in the Web applications in order to retrieve injection points, an injection module which analyzes Web pages, input points to inject attack vectors (such as SQL Injection), and an analysis module to determine possible vulnerabilities based on the system response after attack vector injection. As shown in recent comprehensive studies [16,17], corroborated by research papers [4,5] and confirmed by our own experience with tools such as IBM AppScan[3], these tools suffer from two major weaknesses that highly decrease their practical usefulness:

- **Limitations in application discovery** As black-box Web vulnerability scanners ignore any request that can change the state of the Web applications, they miss large parts of the application. Therefore, these tools test generally a small part of the Web applications due to the ignorance of the application behavioral "intelligence". Due to the growing complexity of the Web applications, they have trouble dealing with specific issues such as infinite Web sites with random URL-based session IDs or automated form submission.
- **Generation of many false positive results** The already-mentioned benchmark shows that a common drawback of these tools is the generation of false positives at a very important rate either for Reflected XSS, SQL Injection or Remote File Inclusion vulnerabilities. The reason is that these tools use brute force mechanisms to fuzz the input data in order to trigger vulnerabilities and establish a verdict by comparison to a reference execution trace. Therefore, they lack precision to assign the verdict, as they do not compute the topology of the Web applications to precisely know where to observe.

These strong limitations of existing Web vulnerability scanners lead to the key objectives of model-based vulnerability testing techniques: better accuracy in vulnerability detection, both by better covering the application (by capturing the behavioral intelligence) and by increasing the precision of the verdict assignment.

[3] `http://www.ibm.com/software/awdtools/appscan/`

In this way, model-based security testing are emerging techniques aiming to leverage model-based approaches for security testing [18]. This includes:

- **Model-based test generation from security protocol, access-control or security-oriented models.** Various types of models of security aspects of the SUT have been considered as input to generate security test. For example, [19] proposes a method using security protocol mutation to infer security test cases. [20] develops a model-based security test generation approach from security models in UMLSec. [21] presents a methodology to exploit a model describing a Web application at the browser level to guide a penetration tester in finding attacks based on logical vulnerabilities.
- **Model-based fuzzing.** This approach applies fuzzing operator in conjunction with models. *Fuzzing techniques* relate to the massive injection of invalid or atypical data (for example by randomly corrupting an XML file) generally by using a randomized approach [22]. Test execution results can therefore expose various invalid behaviors such as crash effects, failing built-in code assertions or memory leaks. [23] proposes an approach that generates invalid message sequences instead of invalid input data by applying behavioral fuzzing operators to valid message UML sequence diagrams.
- **Model-based test generation from weakness or attack models.** Test cases are generated using threat, vulnerability or attacker models, which reflects the attack steps and the required associated data. For example, in [24], threats of security policies modeled with UML sequence diagrams allow to extract event sequences that should not occur during the system execution.

Complementary to these model-based techniques for security testing, our Risk-Based Vulnerability Testing approach is based on a model that captures functional behavioral features of the SUT, but also specifies the fields that allow possible attacks. This feature enables to generate more accurate test cases. Moreover, contrary to functional MBT, the proposed RBVT process is directly driven by the risk analysis (with CORAS) and the vulnerability test patterns, so that the behavioral model is restricted to the only elements that are needed to compute risk-based vulnerability test cases.

5 Conclusion and Future Works

This paper has introduced the RBVT approach, that integrates techniques addressing both risk-based test identification and test prioritization to drive the overall model-based testing generation process. System requirements are used to write the UML test model, while the CORAS security model (in relation with associated generic test pattern catalogue) enables to define selected test purposes and to prioritize them regarding risk assessment. The UML test model completed with selected test purposes defines the input of the test generation tool *CertifyIt*, which automatically derives abstract risk-based vulnerability test cases and next test scripts that can be executed on the SUT. The overall process ensures the traceability between the generated test cases and the targeted vulnerabilities identified during risk assessment.

To achieve and automate this process, we have developed and extended the existing MBT toolchain, based on *CertifyIt*, in order to manage the risk treatment by applying appropriate testing strategies regarding risk assessment. Concretely, the generation of test cases is driven by the risk assessment results, in terms of system perimeter, type of vulnerabilities and associated risk level.

The future work leads in three main research directions: (1) extending the method by covering more vulnerability classes, both technical (such as CSRF, file disclosure and file injection) and logical (such as the integrity of data over applications business processes). We will also (2) investigate methods to gather and aggregate test results, which could be used to automatically complement the risk assessment picture. This feature is indeed enabled by the risk traceability matrix from/to generated test cases and vulnerability objectives. Finally, we want to (3) study the scalability of the testing process to address large scale systems. To reach this goal, we propose to define and support a compositional testing approach by proposing a model composition strategy.

Acknowledgements. The evolution of Smartesting technology to support this risk-based vulnerability test generation is mostly developed within the European FP7 project RASEN[4], which aims to provide risk-based security testing techniques for large-scale networked systems.

References

1. Wichers, D.: Owasp top 10 (October 2013), `https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project` (last visited: February 2014)
2. MITRE: Common weakness enumeration (October 2013), `http://cwe.mitre.org/` (last visited: February 2014)
3. Whitehat: Website security statistics report (October 2013), `https://www.whitehatsec.com/assets/WPstatsReport_052013.pdf` (last visited: February 2014)
4. Doupé, A., Cova, M., Vigna, G.: Why Johnny can't pentest: An analysis of black-box web vulnerability scanners. In: Kreibich, C., Jahnke, M. (eds.) DIMVA 2010. LNCS, vol. 6201, pp. 111–131. Springer, Heidelberg (2010)
5. Finifter, M., Wagner, D.: Exploring the relationship between web application development tools and security. In: Proc. of the 2nd USENIX Conference on Web Application Development (WebApps 2011), Portland, OR, USA, pp. 99–111. USENIX Association (June 2011)
6. Utting, M., Legeard, B.: Practical Model-Based Testing - A tools approach. Morgan Kaufmann, San Francisco (2006)
7. Dias-Neto, A., Travassos, G.: A Picture from the Model-Based Testing Area: Concepts, Techniques, and Challenges. Advances in Computers 80, 45–120 (2010), ISSN: 0065-2458
8. Lebeau, F., Legeard, B., Peureux, F., Vernotte, A.: Model-Based Vulnerability Testing for Web Applications. In: Proc. of the 4th Int. Workshop on Security Testing (SECTEST 2013), Luxembourg, pp. 445–452. IEEE CS Press (March 2013)

[4] `http://www.rasenproject.eu/`

9. Bouquet, F., Grandpierre, C., Legeard, B., Peureux, F.: A test generation solution to automate software testing. In: Proc. of the 3rd Int. Workshop on Automation of Software Test (AST 2008), Leipzig, Germany, pp. 45–48. ACM Press (May 2008)
10. Bouquet, F., Grandpierre, C., Legeard, B., Peureux, F., Vacelet, N., Utting, M.: A subset of precise UML for model-based testing. In: Proc. of the 3rd Int. Workshop on Advances in Model-Based Testing (AMOST 2007), London, UK, pp. 95–104. ACM Press (July 2007)
11. Botella, J., Bouquet, F., Capuron, J.F., Lebeau, F., Legeard, B., Schadle, F.: Model-Based Testing of Cryptographic Components – Lessons Learned from Experience. In: Proc. of the 6th Int. Conference on Software Testing, Verification and Validation (ICST 2013), Luxembourg, pp. 192–201. IEEE CS (March 2013)
12. Bach, J.: Risk and Requirements-Based Testing. Computer 32(6), 113–114 (1999)
13. Lund, M.S., Solhaug, B., Stølen, K.: Model-Driven Risk Analysis: The CORAS Approach, 1st edn. Springer Publishing Company, Incorporated (2010)
14. Vouffo Feudjio, A.G.: Initial Security Test Pattern Catalog. Public Deliverable D3.WP4.T1, Diamonds Project, Berlin, Germany (June 2012) http://publica.fraunhofer.de/documents/N-212439.html (last visited: February 2014)
15. Bau, J., Bursztein, E., Gupta, D., Mitchell, J.: State of the Art: Automated Black-Box Web Application Vulnerability Testing. In: Proc. of the 31st Int. Symp. on Security and Privacy (SP 2010), Oakland, CA, USA, pp. 332–345. IEEE CS (May 2010)
16. Allan, D.: Web application security: automated scanning versus manual penetration testing. IBM White Paper (2008) ftp://ftp.software.ibm.com/software/rational/web/whitepapers/r_wp_autoscan.pdf (last visited: February 2014)
17. SecToolMarket: Price and Feature Comparison of Web Application Scanners (February 2014), http://www.sectoolmarket.com/ (last visited: February 2014)
18. Schieferdecker, I., Grossmann, J., Schneider, M.: Model-based security testing. In: Proc. of the 7th Int. Workshop on Model-Based Testing (MBT 2012), Tallinn, Estonia. EPTCS, vol. 80, pp. 1–12. Open Publishing Association (March 2012)
19. Dadeau, F., Héam, P.-C.: Kheddam, R.: Mutation-Based Test Generation from Security Protocols in HLPSL. In: Proc. of the 4th Int. Conf. on Software Testing, Verification and Validation, Berlin, Germany, pp. 240–248. IEEE CS (March 2011)
20. Jürjens, J.: Model-based Security Testing Using UMLsec: A Case Study. The Journal of Electronic Notes in Theoretical Computer Science (ENTCS) 220(1), 93–104 (2008)
21. Buchler, M., Oudinet, J., Pretschner, A.: Semi-Automatic Security Testing of Web Applications from a Secure Model. In: Proc. of the 6th Int. Conference on Software Security and Reliability (SERE 2012), Gaithersburg, MD, USA, pp. 253–262. IEEE CS (June 2012)
22. Takanen, A., De Mott, J., Miller, C.: Fuzzing for Software Security Testing and Quality Assurance. Artech House, Inc., Norwood (2008)
23. Schneider, M., Großmann, J., Tcholtchev, N., Schieferdecker, I., Pietschker, A.: Behavioral Fuzzing Operators for UML Sequence Diagrams. In: Haugen, Ø., Reed, R., Gotzhein, R. (eds.) SAM 2012. LNCS, vol. 7744, pp. 88–104. Springer, Heidelberg (2013)
24. Wang, L., Wong, E., Xu, D.: A threat model driven approach for security testing. In: Proc. of the 3rd Int. Workshop on Software Engineering for Secure Systems (SESS 2007), Minneapolis, MN, USA. IEEE CS (May 2007)

Medical Cyber-Physical Systems
(Track Introduction)

Ezio Bartocci[1], Sicun Gao[2], and Scott A. Smolka[3]

[1] Vienna University of Technology, Austria
[2] Carnegie Mellon University, USA
[3] Stony Brook University, USA

1 Introduction

Rapid progress in modern medical technologies has led to a new generation of health-care devices and treatment strategies. Examples include electro-anatomical mapping and intervention, bio-compatible and implantable devices, minimally invasive embedded devices, and robotic prosthetics.

Medical Cyber-Physical Systems (CPS) refer to modern medical technologies in which sophisticated and highly complex embedded systems equipped with network communication capabilities, are responsible for monitoring and controlling the physical dynamics of patients' bodies. These systems share a key characteristic: the tight integration of digital computation, responsible for control and communication in discrete-time, with a physical system, obeying laws of physics and evolving in continuous-time.

Malfunctioning of these devices can do great harm to human health. The verification, validation and certification of their reliability and safety are extremely important and still very challenging tasks, owing to the complexity of the involved interactions. Research in formal methods is leading to mathematically rigorous techniques for ensuring the correct design and implementation.

The behavior of Medical CPS is characterized by the nonlinear interaction between discrete (computing device) and continuous phenomena (the patient's body). For this reason, research on hybrid systems verification [6,8,11,12,5] plays a key role in analysing such systems. Furthermore, the modelling and the efficient simulation of the patient body is becoming very important for the design and validation of Medical CPS and for the development of personalised treatment strategies.

GPGPUs (General-Purpose Computing on Graphics Processing Units) are increasingly being used to achieve simulation speeds [3,18] in near real-time for complex spatial patterns indicative of cardiac arrhythmic disorders. Real-time simulation of organs without the need for supercomputers may soon facilitate the adoption of model-based clinical diagnostics and treatment planning. This will reduce also the number of experiments and tests, reducing the discomfort for the patient and legal issues.

The availability of software and hardware accelerators (such as FPGA) for real-time verification (monitoring) of signals, makes this approach very attractive also in medicine, for instance to monitor the ECG signal [2] or the flow/pressure curves of assisted ventilation of a patient in intensive care [7]. Formal specification languages, such as Temporal Logics (TL), have proved to be a powerful and natural framework to describe complex temporal properties of systems [10]. In this context, one important research challenge will be to combine machine learning and model checking techniques [14,1] to learn from a training set of data the TL specifications that can better discriminate normal physiological behaviors from critical ones [2].

2 Overview of the Session Papers

The session consisted of five contributed papers. The paper by Grosu et al. [13] offers an insight of a joint US project proposal, that involves several american universities and institutions, on the development of a model-based design framework for medical devices to verify and to test the safety and efficacy of device software for implantable cardiac devices such as pacemakers and defibrillators. In the last twenty years the rate of software failures of all medical devices recalled from the market have more than doubled. Thus, there is a great need of a formal design methodology or open experimental platforms that can be used to ensure the correct operation of medical devices within the physiological closed-loop context.

On the same line of research is the paper by Kwiatkowska et al. [15]. This contribution gives an overview of a model-based framework developed on hybrid automata [4,14] to support a range of quantitative verification techniques [8,16] for the analysis of safety, reliability and energy usage of pacemakers. This framework aims also to provide techniques for parametric analysis of personalised physiological properties to test in silico new implantable device designs on patients, thus reducing the cost and discomfort.

The paper by Clarke et al. [9] presents the research in progress of the authors on model checking of safety-critical hybrid systems involving both discrete and continuous behaviors, such as medical devices of various sorts. Current industrial model checkers do not scale to handle realistic hybrid systems. The key idea proposed in this paper is to handle more complex systems by combining existing discrete methods in model checking with new algorithms based on computable analysis.

The paper by Leucker et al. [17] presents some open research and technical challenges, legal issues and proposed solutions concerning the integration testing required when medical devices produced by different manufacturers need to be interconnected. Medical CPS, especially when operated in the surgery room, are safety critical systems, upon which the patient's life may depend. Hence, the certification of a well-behave integration may have a tremendous economical impact, reducing the life-threatening risks for the patients.

Finally, the paper by Bufo et al. [7] introduces a novel approach to automatically detect ineffective breathing efforts in patients in intensive care subject to assisted ventilation. The method is based on synthesising from data temporal logic formulae which are able to discriminate between normal and ineffective breaths.

References

1. Bartocci, E., Bortolussi, L., Nenzi, L., Sanguinetti, G.: On the robustness of temporal properties for stochastic models. In: Proc. of HSB 2013. EPTCS, vol. 125, pp. 3–19 (2013)
2. Bartocci, E., Bortolussi, L., Sanguinetti, G.: Data-driven statistical learning of temporal logic properties. In: Legay, A., Bozga, M. (eds.) FORMATS 2014. LNCS, vol. 8711, pp. 23–37. Springer, Heidelberg (2014)

3. Bartocci, E., Cherry, E.M., Glimm, J., Grosu, R., Smolka, S.A., Fenton, F.H.: Toward real-time simulation of cardiac dynamics. In: Proc. of CMSB 2011, pp. 103–112. ACM (2011)
4. Bartocci, E., Corradini, F., Di Berardini, M.R., Smolka, S.A., Grosu, R.: Modeling and simulation of cardiac tissue using hybrid I/O automata. Theor. Comput. Sci. 410(33-34), 3149–3165 (2009)
5. Bartocci, E., Corradini, F., Entcheva, E., Grosu, R., Smolka, S.A.: CellExcite: An efficient simulation environment for excitable cells. BMC Bioinformatics 9(suppl. 2), S3 (2008)
6. Bartocci, E., Liò, P., Merelli, E., Paoletti, N.: Multiple verification in complex biological systems: The bone remodelling case study. T. Comp. Sys. Biology 14, 53–76 (2012)
7. Bufo, S., Bartocci, E., Sanguinetti, G., Borelli, M., Lucangelo, U., Bortolussi, L.: Temporal logic based monitoring of assisted ventilation in intensive care patients. In: Margaria, T., Steffen, B. (eds.) ISoLA 2014, Part II. LNCS, vol. 8803, pp. 393–405. Springer, Heidelberg (2014)
8. Chen, T., Diciolla, M., Kwiatkowska, M., Mereacre, A.: Quantitative verification of implantable cardiac pacemakers over hybrid heart models. Inf. and Comp. 236, 87–101 (2014)
9. Clarke, E.M., Gao, S.: Model checking hybrid systems. In: Margaria, T., Steffen, B. (eds.) ISoLA 2014, Part II. LNCS, vol. 8803, pp. 387–388. Springer, Heidelberg (2014)
10. Donzé, A., Maler, O., Bartocci, E., Nickovic, D., Grosu, R., Smolka, S.: On temporal logic and signal processing. In: Chakraborty, S., Mukund, M. (eds.) ATVA 2012. LNCS, vol. 7561, pp. 92–106. Springer, Heidelberg (2012)
11. Gao, S., Kong, S., Clarke, E.M.: dReal: An SMT solver for nonlinear theories over the reals. In: Bonacina, M.P. (ed.) CADE 2013. LNCS (LNAI), vol. 7898, pp. 208–214. Springer, Heidelberg (2013)
12. Grosu, R., Batt, G., Fenton, F.H., Glimm, J., Le Guernic, C., Smolka, S.A., Bartocci, E.: From cardiac cells to genetic regulatory networks. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 396–411. Springer, Heidelberg (2011)
13. Grosu, R., et al.: Compositional, approximate, and quantitative reasoning for medical cyber-physical systems with application to patient-specific cardiac dynamics and devices. In: Chakraborty, S., Mukund, M. (eds.) ISoLA 2014, Part II. LNCS, vol. 8803, pp. 360–365. Springer, Heidelberg (2014)
14. Grosu, R., Smolka, S.A., Corradini, F., Wasilewska, A., Entcheva, E., Bartocci, E.: Learning and detecting emergent behavior in networks of cardiac myocytes. Commun. ACM 52(3), 97–105 (2009)
15. Kwiatkowska, M., Mereacre, A., Paoletti, N.: On quantitative software quality assurance methodologies for cardiac pacemakers. In: Chakraborty, S., Mukund, M. (eds.) ISoLA 2014, Part II. LNCS, vol. 8803, pp. 366–386. Springer, Heidelberg (2014)
16. Kwiatkowska, M., Lea-Banks, H., Mereacre, A., Paoletti, N.: Formal modelling and validation of rate-adaptive pacemakers. In: IEEE International Conference on Healthcare Informatics, ICHI 2014 (to appear, 2014)
17. Leucker, M.: Challenges for the dynamic interconnection of medical devices. In: Chakraborty, S., Mukund, M. (eds.) ISoLA 2014, Part II. LNCS, vol. 8803, pp. 389–392. Springer, Heidelberg (2014)
18. Murthy, A., Bartocci, E., Fenton, F., Glimm, J., Gray, R., Cherry, E., Smolka, S., Grosu, R.: Curvature analysis of cardiac excitation wavefronts. IEEE/ACM Transactions on Computational Biology and Bioinformatics 10(2), 323–336 (2013)

Compositional, Approximate, and Quantitative Reasoning for Medical Cyber-Physical Systems with Application to Patient-Specific Cardiac Dynamics and Devices

Radu Grosu[1], Elizabeth Cherry[2], Edmund M. Clarke[3], Rance Cleaveland[4], Sanjay Dixit[5], Flavio H. Fenton[6], Sicun Gao[3], James Glimm[1], Richard A. Gray[7], Rahul Mangharam[5], Arnab Ray[8], and Scott A. Smolka[1]

[1] Stony Brook University
[2] Rochester Institute of Technology
[3] Carnegie Mellon University
[4] University of Maryland
[5] University of Pennsylvania
[6] Georgia Institute of Technology
[7] U.S. Food and Drug Administration
[8] Fraunhofer USA Center for Experimental Software Engineering

Abstract. The design of bug-free and safe medical device software is challenging, especially in complex implantable devices that control and actuate organs who's response is not fully understood. Safety recalls of pacemakers and implantable cardioverter defibrillators between 1990 and 2000 affected over 600,000 devices. Of these, 200,000 or 41%, were due to firmware issues that continue to increase in frequency. According to the FDA, software failures resulted in 24% of *all* medical device recalls in 2011. There is currently no formal methodology or open experimental platform to test and verify the correct operation of medical-device software within the closed-loop context of the patient.

The goal of this effort is to develop the foundations of modeling, synthesis and development of *verified medical device software* and systems *from verified closed-loop models* of the device and organ(s). Our research spans both implantable medical devices such as cardiac pacemakers and physiological control systems such as drug infusion pumps which have multiple networked medical systems. These devices are physically connected to the body and exert direct control over the physiology and safety of the patient. The focus of this effort is on (a) Extending current binary safety properties to quantitative verification; (b) Development of patient-specific models and therapies; (c) Multi-scale modeling of complex physiological phenomena and compositional reasoning across a range of model abstractions and refinements; and (d) Bridging the formal reasoning and automated generation of safe and effective software for future medical devices.

1 Introduction

Between 1992-1998, less than 10% of medical devices were recalled due to software issues. This rate has more than doubled in 2011 with software failures accounting for

24% of all medical device recalls. There is currently no formal design methodology or open experimental platforms that can be used to ensure the correct operation of medical devices *within the physiological closed-loop context*. Furthermore, the present approach of ad hoc and open-loop testing of medical device software and the design process significantly increase the time and cost for validation and do not provide strong guarantees on the safety and efficacy of the closed-loop system of the device and the patient. Given the increasing complexity and features built in medical devices, the rate and volume of devices recalled will continue on its current trajectory, unless a systematic approach for medical device software verification, validation and testing, within clinical and physiological relevant contexts, is adopted.

The focus of this proposal is on the development of a model-based design framework for medical devices to verify and test the safety and efficacy of device software for implantable cardiac devices such as pacemakers and defibrillators. This will be accomplished in three phases:

(a) Integrated Functional and Formal Modeling: We propose a multi-scale modeling approach where abstract physiological and device models are used to prove basic safety closed-loop properties and progressively refined models automatically prove more complex properties. We are particularly interested in cases where the device may drive the heart into unsafe states, such as in Pacemaker Mediated Tachycardia. To accomplish this, we will develop approximate and probabilistic physiological models for quantitative verification for competitive analysis of new cardiac rhythm therapies.

(b) Patient-specific Modeling: Using the generalized modeling approaches we will now employ patient data to develop patient-specific tuned heart models and conduct sensitivity and parametric analysis for model-based clinical trials of implantable cardiac devices.

(c) Pre-Clinical Validation and Platforms The modeling effort will be directed and supported by clinical validation with evaluation of therapies on animal models and organs. The heart and device models and the therapies developed in this effort will be implemented in closed-loop testing platforms to standardize the toolchains for low-cost and efficient medical device software evaluation. With collaboration with the US FDA, the proposed framework, models, platforms and toolchain will be harmonized into the current regulatory guidelines for development of high-confidence medical device software and systems.

1.1 From Verified Models to Verified Code for Medical Devices

Model-based approaches are revolutionizing the development of cyber-physical systems in general, and embedded control systems in particular. In these paradigms, which are variously called Model-Based Development (MBD) or Model-Driven Engineering (MDE), engineers first build models of the components of the system under development. They then use simulations to verify that the system exhibits desired properties [3, 4, 9, 17, 19], synthesis techniques ("autocoding") to generate portions of the implementation automatically, and hybrid simulation/hardware-test infrastructure ("hardware-in-the-loop testing") to verify implementations of components as they become available. The motivations for MBD/MDE approaches stem from time and cost efficiencies in engineering processes: the “virtual prototyping” enabled by computer-based modeling permits much more thorough analysis of a design, at much lower cost, than does traditional physical prototyping.

The use of MBD/MDE is especially advanced in the automotive and aerospace control domains, where detailed simulation models for the physics of controlled systems (“plants”) have been developed and serve as the basis for assessing models of control strategies proposed by engineers building these vehicles. In other domains, such as medical-device design, these techniques have yet to achieve much headway, due in part to a lack widely accepted behavioral models for human biological systems, but also due to the wide variability observed in individual patient’s biological functions.

The goals of this proposal are to develop the theoretical and practical underpinnings of a new verification framework for cyber-physical systems that would support compositional, highly parameterizable, approximate, and quantitative reasoning; to build the tool support for conducting the deep analysis this framework will allow; and to use these tools and techniques to advance the state of the art in cardiac therapy devices. We are particularly interested in *closed-loop verification* of cardiac device software and therapies [15, 19]. In this setting, a computational model of the heart (the biological plant) is under closed-loop control of a computational model of the cardiac device (the controller), and verification is conducted on this closed-loop system. Moreover, we will develop a *multi-scale* formal modeling approach, in which simpler properties are verified using more abstract models for the heart and device, while more complex properties require progressively refined plant and controller models.

We are also very interested in developing *patient-specific* heart models, which we plan to obtain from ablation and other cardiac-related procedures. Having such models in the verification loop will improve the level of confidence in the safety and efficacy of the device, thereby potentially reducing the expense of failed clinical trials.

An architectural overview of our proposed framework, which we call HYRES, is given in Figure 1.[1] In what follows, we summarize our proposed work on the verification technology needed to support closed-loop verification of medical CPSs, its application to cardiac devices, especially the recent proposed Low-Energy Anti-Fibrillatory Pacing (LEAP) [16] approach of PIs Fenton and Cherry and its interaction with more traditional pacing and anti-arrhythmic therapies and our planned education and outreach activities.

2 Computational Foundations for Medical CPSs

Compositional Reasoning (Plug and Play). If two components (subsystems) are approximately equivalent (they can simulate each other’s behavior up to a small error δ), then it would be highly desirable if you could replace one with the other in a larger context in a guaranteed *safe* manner; i.e., the resulting total behaviors are approximately equivalent up to a small error ϵ, which is a function of δ. For this to be the case, one needs to provide appropriate *proof rules* (based most likely on a *small-gain condition*), since in most interesting cases the larger context is nonlinear.

[1] The name HYRES derives from *Hybrid systems*, a modeling formalism for CPSs amenable to formal verification, and the *Resolution* or precision at which the verification is carried out.

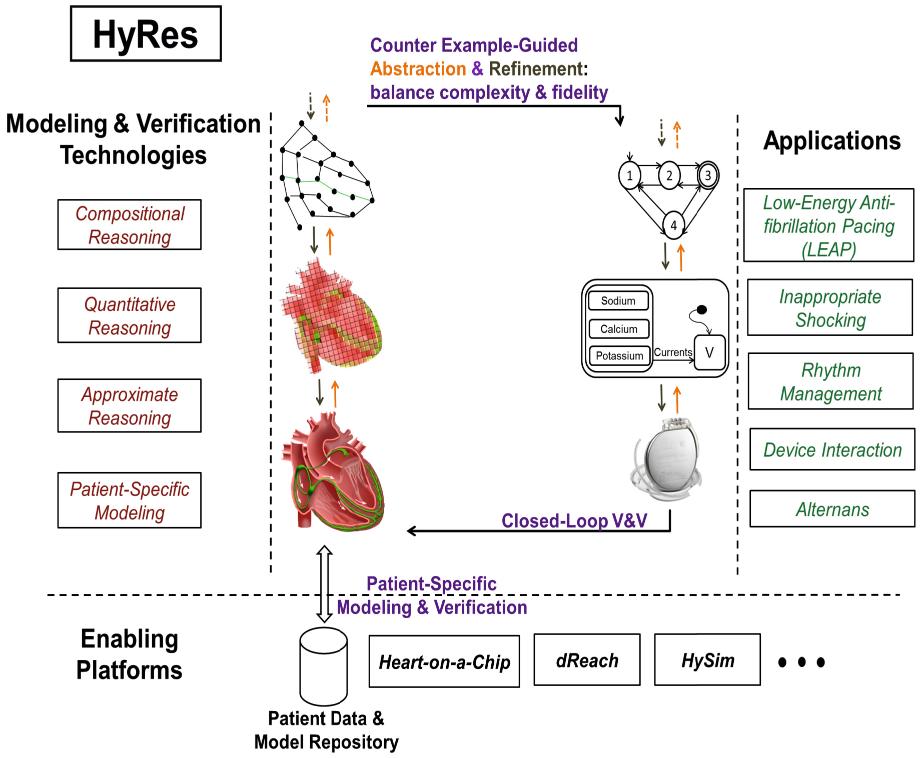


Fig. 1. The HyRes framework for closed-loop verification of Medical CPS. The verification technologies we propose to develop are shown on the left, the intended applications on the right, and the supporting computational platforms and repositories along the bottom of the figure. A hierarchy of models, capturing the electrophysiology of the heart at varying levels of complexity, will be devised using abstraction and refinement techniques. The figure shows a highly detailed model at the base of the hierarchy, which is spatially abstracted to obtain a grid-based computational model, which is further abstracted to obtain a network of Timed automata for reasoning about timing-related properties.

PIs Grosu, Smolka and others have recently used this kind of reasoning to show that the 13-variable sodium-channel component of the 67-variable IMW cardiac-cell model (Iyer-Mazhari-Winslow) can be replaced by an approximately bisimilar, 2-variable HH-type (Hodgkin-Huxley) abstraction [12–14, 18] . Moreover, this substitution of (approximately) equals for equals is safe in the sense that the approximation error between sodium-channel models is not amplified by the feedback-loop context in which it is placed.

Being able to reason about dynamical systems compositionally [1, 13, 14] is important for two reasons: the plug-and-safely-play nature of compositional reasoning is highly *efficient*, as it avoids the state-explosion problem that bedevils automated verification; and composition of subsystems can be used to uncover *bad interactions* between subsystems, AKA the *feature interaction* problem.

Compositional reasoning can also be used as basis for *synthesis*: not just controller synthesis (well-studied), but plant synthesis. That is, given a controller, infer the plant for which it works.

Approximate Logical Reasoning (Maximum Precision). When proving that a system satisfies a particular property, there is a “Plank discretization” (precision) limit. For example, suppose a curve (given by an analytic function) separates the plane, and that there is a small grid of “Plank size”. For the grid squares cut by the (zero-width) curve, we cannot say whether they are on one side or the other of the curve (that is, satisfy or do not satisfy the property). For all other squares, one has a definitive answer.

Approximate verification can also be used to turn an undecidable decision problem over the reals into a decidable decision problem, and efficiently at that. In a multi-time-scale approach to verification, choose the level of approximation that matches the granularity of the time-scale under consideration.

The team brings expertise in this approach to the proposed effort in the form of the dReal and dReach *reachability analysis platform for nonlinear hybrid systems*, cultivated by PIs Gao and Clarke during the course of the CMACS NSF Expedition in Computing. In the spirit of this proposal, Gao and Clarke have applied dReal/dReach to the analysis of a highly nonlinear cardiac-cell model [7–9].

Quantitative Logical Reasoning (How Good). Classical temporal-logic model checking provides a boolean yes/no answer to the question “Does a system Σ satisfy a temporal logic formula φ?” When Σ is a dynamical system such as a CPS, one can demand a more quantitative assessment of how well Σ does or does not satisfy φ. If φ is satisfied by Σ, then how *robustly* is it satisfied? If φ is violated, then how badly is it violated? How many (abstract) points in the state space violate the property? If a point violates the property, then by how much does it violate it? Quantitative reasoning [2, 10, 11] can be seen as lifting the model checking problem from a boolean setting to one in which the results are interpreted over a metric space.

With quantitative reasoning, once can also augment temporal logic with *quantitative operators*. For example, consider the following *convergence* property $FG(x \leq \tau)$, which states that eventually the value of x is always less than or equal to threshold τ. In the quantitative setting, one can also measure the *speed* at which the G-subformula eventually becomes true, and the *average* value of x, once x always $\leq \tau$.

Quantitative reasoning can also play a role *diagnostically*. Consider the safety property: an ICD should not deliver an inappropriate shock, or the occurrence of one should be minimized. Quantitatively, one can compute e.g. the average amount of energy consumed by an ICD every time an inappropriate shock is delivered to the patient.

Adversarial Reasoning (Games, and Open Systems). The controller and the plant do not always represent a closed system. They may be in a game-like situation with the environment from which they receive additional *adversarial* input. A winning strategy for the controller + system is one for which they behave *safely* regardless of the the moves the environment makes. The environment may be nondeterministic (making it difficult to compete against), or stochastic (making it somewhat easier to deal with as one can then model it with belief states and partially observable Markov decision processes).

Closed-loop Verification with Automated Model Abstraction and Refinement. While complex physiological models of the heart with over 4 million finite elements or 100K ODEs exist, they do not provide a suitable level of interaction with a device such as a pacemaker which only observes the state of the heart from two or three points. We propose a multi-scale formal modeling approach to verify a set of closed-loop properties (i.e., where the heart can affect the device and, more importantly, where the device can drive the heart into safe/unsafe states).

In this approach, simpler properties are verified with more abstract models of the heart/device, while more complex properties require progressively refined plant models. In support of this approach, we will develop automated a Counter-Example-Guided Abstraction and Refinement (CEGAR) framework to balance model complexity and fidelity in accordance with increasingly complex closed-loop issues such as Pacemaker Mediated Tachycardia, where the pacemaker drives the heart into an unsafe state.

3 Application to Patient-Specific Cardiac Models, Therapies, and Devices

Patient-Specific Modeling The construction of patient-specific heart models will enable:

- Improved level of confidence in the safety and efficacy of the device with a patient-model in the loop. This will reduce the expense of failed clinical trials and potentially reduce the extent of clinical trials, in general.
- Physicians to maintain actionable patient records between operations and perform pre-op evaluations on these models.
- Semi-automatic tuning of device parameters to the specific patient requirements.
- Model-based training of EP fellows and medical students.

The requisite data will be obtained from ablation and other medical, cardiac-related procedures. We will use this data to learn/personalize heart models, device settings, etc. We refer to this process as *Patent-Specific (P-S) Modeling*. In order to have P-S heart models, we will need to incorporate patient data in our models and tool chain. The most accurate patient data comes from the electro-physiology study before implantation. Catheters with probes are inserted into the patient's heart and local electrical activities are recorded as Electrogram (EGM) signals. From the EGM signals, we can extract timing delays between different heart locations. As the VHM and EP studies use the same parameters, we can incorporate patient EGM data to form a P-S heart model. We will pursue this in two steps:

(1) Model Construction Using Synthetic EGMs: The VHM is able to generate synthetic EGM signals. Since EGM signals mainly carry timing information, the synthetic EGMs are comparable to realistic EGMs - however with known probability distributions. As the VHM is a more controlled environment than a real patient, it is much

easier to evaluate for quantitative verification for patient-specific conditions.
(2) Model Construction using Realistic EGMs: We will use EGMs from a real patient to construct our model. This will require noise filtering, determining the catheter positioning and benchmark analysis for the constructed model.

We are also interested in *Property-Based Modeling*. If one is only interested in timing-related aspects of patient therapy, as may be the case with a pacemaker, learn a Timed Automaton (TA) model of the patient's heart and of the device. If voltage is of interest, for example in the treatment of VT and VFib, learn a voltage-based Hybrid Automaton (HA) model.

A key aspect of property-based modeling will be to ensure that the models we derive are related to one another in the ways we intend them to be. For example, is the TA model an abstraction of the HA model? We can ensure this is the case by following a process of *abstraction refinement* in deriving e.g. the HA model from the TA one.

Such a framework will allow for enforcement of property priorities, where under certain physiological conditions, some properties may be violated while higher-priority properties remain enforced. This will allow for verification of multi-scale and multi-mode systems, whose properties must adapt to the mode of the patient.

Closed-Loop Verification of Cardiac Therapies and their Interactions. We will put a P-S cardiac model in the loop with a cardiac device with P-S parameter settings, and apply the analysis techniques developed in Part I of the proposal to the resulting systems. Recent work by the PIs in compositional verification [12–14, 18] (Grosu and Smolka), approximate verification [5–8] (Clarke and Gao), and quantitative reasoning [10] (Grosu, Smolka, and others) on which this proposal will build makes us confident that we will be successful.

We will consider both pacemakers and ICDs and their interactions. Some devices combine a pacemaker and ICD in one unit for persons who need both functions, and this is becoming more and more common. Thus, the need to carefully analyze their interactions is on the rise.

Low-Energy Anti-Fibrillatory Pacing (LEAP). PIs Fenton, Cherry, and others have developed a new approach to eradicating life-threatening arrhythmias. Instead of one large jolt of electricity to the heart, the new approach, called low-energy anti-fibrillatory pacing (LEAP), uses a series of smaller electrical pulses. An article describing this breakthrough appeared in a recent issue of Nature [16]. The goal of LEAP is not to eliminate the arrhythmia at once, but rather to synchronize the electrical state of the heart gradually. In this way, undesirable side effects can be avoided while still restoring the heart to its normal condition.

Computational modeling, initially using simple models and then more complex models [3, 17], validated this approach and provided guidance for a series of preclinical experimental trials that demonstrated LEAP's effectiveness. The modeling and analysis techniques put forth in this proposal will be used to further optimize the method so that it can be used in human clinical trials. We will also study its interactions with other pacing-based therapies.

References

1. Bartocci, E., Bortolussi, L., Nenzi, L.: A temporal logic approach to modular design of synthetic biological circuits. In: Gupta, A., Henzinger, T.A. (eds.) CMSB 2013. LNCS (LNBI), vol. 8130, pp. 164–177. Springer, Heidelberg (2013)
2. Bartocci, E., Bortolussi, L., Nenzi, L., Sanguinetti, G.: On the robustness of temporal properties for stochastic models. In: Proc. of HSB 2013: The 2nd Intern. Workshop on Hybrid Systems and Biology. EPTCS, vol. 125, pp. 3–19 (2013)
3. Bartocci, E., Cherry, E.M., Glimm, J., Grosu, R., Smolka, S.A., Fenton, F.H.: Toward real-time simulation of cardiac dynamics. In: Proceedings of the 9th International Conference on Computational Methods in Systems Biology, CMSB 2011, pp. 103–112. ACM, New York (2011)
4. Bartocci, E., Singh, R., von Stein, F.B., Amedome, A., Caceres, A.J., Castillo, J., Closser, E., Deards, G., Goltsev, A., Ines, R.S., Isbilir, C., Marc, J.K., Moore, D., Pardi, D., Sadhu, S., Sanchez, S., Sharma, P., Singh, A., Rogers, J., Wolinetz, A., Grosso-Applewhite, T., Zhao, K., Filipski, A.B., Gilmour, R.F., Grosu, R., Glimm, J., Smolka, S.A., Cherry, E.M., Clarke, E.M., Griffeth, N., Fenton, F.H.: Teaching cardiac electrophysiology modeling to undergraduate students: Laboratory exercises and GPU programming for the study of arrhythmias and spiral wave dynamics. Adv. Physiol. Educ. 35(4), 427–437 (2011)
5. Gao, S., Avigad, J., Clarke, E.M.: Delta-complete decision procedures for satisfiability over the reals. In: Proceedings of the 6th International Joint Conference on Automated Reasoning (IJCAR), pp. 286–300 (2012)
6. Gao, S., Avigad, J., Clarke, E.M.: Delta-decidability over the reals. In: Proceedings of the 27th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), pp. 305–314 (2012)
7. Gao, S., Kong, S., Clarke, E.M.: dReal: An SMT solver for nonlinear theories over the reals. In: Bonacina, M.P. (ed.) CADE 2013. LNCS (LNAI), vol. 7898, pp. 208–214. Springer, Heidelberg (2013)
8. Gao, S., Kong, S., Clarke, E.M.: Satisfiability modulo ODEs. In: Proceedings of the 13th International Conference on Formal Methods in Computer Aided Design, FMCAD (2013)
9. Grosu, R., Batt, G., Fenton, F.H., Glimm, J., Le Guernic, C., Smolka, S.A., Bartocci, E.: From cardiac cells to genetic regulatory networks. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 396–411. Springer, Heidelberg (2011)
10. Grosu, R., Peled, D., Ramakrishnan, C.R., Smolka, S.A., Stoller, S.D., Yang, J.: Compositional branching-time measurements. In: Bensalem, S., Lakhneck, Y., Legay, A. (eds.) FPS 2014 (Sifakis Festschrift). LNCS, vol. 8415, pp. 118–128. Springer, Heidelberg (2014)
11. Grosu, R., Peled, D., Ramakrishnan, C.R., Smolka, S.A., Stoller, S.D., Yang, J.: Using statistical model checking for measuring systems. In: Margaria, T., Steffen, B. (eds.) ISoLA 2014, Part II. LNCS, vol. 8803, pp. 223–238. Springer, Heidelberg (2014)
12. Islam, M.A., Murthy, A., Bartocci, E., Girard, A., Smolka, S., Grosu, R.: Compositionality results for cardiac cell dynamics. In: Gupta, A., Henzinger, T.A. (eds.) CMSB 2013. LNCS, vol. 8130, pp. 242–244. Springer, Heidelberg (2013)
13. Islam, M.A., Murthy, A., Bartocci, E., Cherry, E.M., Fenton, F.H., Glimm, J., Smolka, S.A., Grosu, R.: Model-order reduction of ion channel dynamics using approximate bisimulation. Theoretical Computer Science (in press, 2014)
14. Islam, M.A., Murthy, A., Girard, A., Smolka, S.A., Grosu, R.: Compositionality results for cardiac cell dynamics. In: Proc. of HSCC 2014: The 17th International Conference on Hybrid Systems: Computation and Control, HSCC 2014, pp. 243–252. ACM, New York (2014)
15. Jiang, Z., Pajic, M., Alur, R., Mangharam, R.: Closed-loop verification of medical devices with model abstraction and refinement. STTT 16(2), 191–213 (2014)

16. Luther, S., Fenton, F.H., Kornreich, B.G., Squires, A., Bittihn, P., Hornung, D., Zabel, M., Flanders, J., Gladuli, A., Campoy, L., Cherry, E.M., Luther, G., Hasenfuss, G., Krinsky, V.I., Pumir, A., Gilmour, R.F., Bodenschatz, E.: Low-energy control of electrical turbulence in the heart. Nature 475(7355), 235–239 (2011)
17. Murthy, A., Bartocci, E., Fenton, F., Glimm, J., Gray, R., Cherry, E., Smolka, S., Grosu, R.: Curvature analysis of cardiac excitation wavefronts. IEEE/ACM Transactions on Computational Biology and Bioinformatics 10(2), 323–336 (2013)
18. Murthy, A., Islam, M.A., Bartocci, E., Cherry, E.M., Fenton, F.H., Glimm, J., Smolka, S.A., Grosu, R.: Approximate bisimulations for sodium channel dynamics. In: Gilbert, D., Heiner, M. (eds.) CMSB 2012. LNCS, vol. 7605, pp. 267–287. Springer, Heidelberg (2012)
19. Pajic, M., Jiang, Z., Lee, I., Sokolsky, O., Mangharam, R.: From verification to implementation: A model translation tool and a pacemaker case study. In: Proceedings of IEEE 18th Real Time and Embedded Technology and Applications Symposium, Beijing, China, April 16-19, pp. 173–184 (2012)

On Quantitative Software Quality Assurance Methodologies for Cardiac Pacemakers

Marta Kwiatkowska, Alexandru Mereacre, and Nicola Paoletti

Department of Computer Science, University of Oxford, UK

Abstract. Embedded software is at the heart of implantable medical devices such as cardiac pacemakers, and rigorous software design methodologies are needed to ensure their safety and reliability. This paper gives an overview of ongoing research aimed at providing software quality assurance methodologies for pacemakers. A model-based framework has been developed based on hybrid automata, which can be configured with a variety of heart and pacemaker models. The framework supports a range of quantitative verification techniques for the analysis of safety, reliability and energy usage of pacemakers. It also provides techniques for parametric analysis of personalised physiological properties that can be performed *in silico*, which can reduce the cost and discomfort of testing new designs on patients. We describe the framework, summarise the results obtained, and identify future research directions in this area.

Keywords: model-based design; quantitative verification; hybrid automata; heart modelling; cardiac pacemakers.

1 Introduction

The growing reliance on implantable medical devices controlled by embedded software calls for rigorous software design methodologies to ensure their safe operation and to avoid costly device recalls. We focus here on cardiac pacemakers, which are battery-powered devices implanted under a patient's skin that sense the electrical signals in the heart and regulate the heart rhythm. Of paramount importance here is the safety of the device's operation, but analysis of characteristics such as energy usage are also needed to improve the designs. An important observation is that evaluating the operation of the pacemaker must take into account the characteristics of the heart rhythm of the patient, and therefore personalisation of the methodology is desirable.

Several models for pacemakers have been proposed, to mention [10, 16, 17, 19, 21, 24, 25]. Since the basic function of the pacemaker is to maintain a normal heart rhythm of 60-100 beats per minute (BPM), the models need to capture real-time, in addition to being able to sense electrical signals, typically (non-linear) continuous flows. Therefore, natural models for the pacemaker are (deterministic) timed or hybrid automata, which are then composed with a heart model, typically a hybrid automaton, for the analysis. An important consideration in our work has been *stochasticity*, which manifests itself in several ways:

sensor noise, modulated rate in response to activity level, as well as the randomness in the timing of the heart beats, which is specific to the patient and can switch between normal and diseased behaviours. We have thus concentrated our efforts on developing effective methodologies to provide software quality assurance for pacemakers in presence of stochasticity through *quantitative verification* techniques.

This paper reports on a comprehensive model-based framework to provide software quality assurance for cardiac pacemakers developed within the VERIWARE and VERIPACE projects and described in [4–6, 18]. The framework is based on hybrid input-output automata models, and can be instantiated with a number of heart models, including a model based on synthetic ECG that can be learnt from patient data and a physiologically-relevant heart model built as a network of cardiac cells. Models of pacemakers of differing functionalities can be plugged into the framework for analysis: we consider a basic pacemaker design inspired by [17], an advanced design that can handle pacemaker mediated tachycardia, as well as a rate-adaptive pacemaker. We implement the framework in Simulink and provide a broad range of analysis techniques, which are based on simulation, as well as approximate quantitative verification, for checking safety, reliability and detailed energy-usage. We also develop analysis methods for advanced physiological properties, including pacemaker mediated tachycardia correction and parametric analysis to support *in silico* testing of the rate-modulation functionality under different personalised scenarios, e.g., age of the patient and activity level. We demonstrate the usefulness of our methodology through a range of experiments. Finally, we summarise future research directions and challenges in this area.

2 Model-based Framework for the Verification of Pacemakers

Our framework for modelling and quantitative verification of pacemaker models is based on the formalism of *hybrid input-output automata* [20], and supports the composition of a heart model and a pacemaker model on which verification is performed. We consider a discrete-time simulation semantics, which enables a sound and straightforward encoding of the formal specification into MATLAB Simulink/Stateflow models. In the following, we recall the basic details of the framework that we introduced in [6].

Let $\mathcal{X} = \{x_1, \ldots, x_d\}$ be a set of variables in $\mathbb{R}$. An $\mathcal{X}$-valuation is a function $\eta : \mathcal{X} \to \mathbb{R}$ assigning to each variable $x \in \mathcal{X}$ a real value $\eta(x)$. Let $\mathcal{V}(\mathcal{X})$ denote the set of all valuations over $\mathcal{X}$. A *constraint* on $\mathcal{X}$, denoted by *grd*, is a conjunction of expressions of the form $x \bowtie c$ for variable $x \in \mathcal{X}$, comparison operator $\bowtie \in \{<, \leq, >, \geq\}$ and $c \in \mathbb{R}$. Let $\mathcal{B}(\mathcal{X})$ denote the set of constraints over $\mathcal{X}$. Let $\mathcal{Y}(\mathcal{X})$ denote the set of all real-valued functions over $2^{\mathcal{X}}$. We define $\mathcal{L}(\mathcal{X}) := \{x := u \mid x \in \mathcal{X} \wedge u \in \mathcal{X} \cup \{0\}\}$ to be the set of *update* assignments over the set of variables $\mathcal{X}$.

Definition 1 (Hybrid I/O Automaton). *A hybrid I/O automaton (HIOA) $\mathcal{A} = (\mathcal{X}, Q, q_0, E_1, E_2, \mathrm{Inv}, \rightarrow, \mathrm{Diff})$ consists of:*

- *a finite set of variables $\mathcal{X}$;*
- *a finite set of modes Q, with the initial mode $q_0 \in Q$;*
- *a finite set E_1 of* input *actions and a finite set E_2 of* output *actions with $\mathcal{E} = E_1 \cup E_2$;*
- *an invariant function $\mathrm{Inv} : Q \rightarrow \mathcal{B}(\mathcal{X})$;*
- *a transition relation $\rightarrow \subseteq Q \times (\mathcal{E} \cup \{\varsigma\}) \times \mathcal{B}(\mathcal{X}) \times 2^{\mathcal{L}(\mathcal{X})} \times Q$, where ζ is the internal action; and*
- *a derivative function $\mathrm{Diff} : Q \times \mathcal{X} \rightarrow \mathcal{Y}(\mathcal{X})$ that assigns a function to a variable $x \in \mathcal{X}$.*

We use a *network of HAs* for the composition of more than one HA. In order to obtain a deterministic network we impose some restrictions on HAs as follows:

- they must be *input enabled*, meaning that, for each mode and each input action, there is an edge labelled by the input action;
- the output actions have the highest priority, meaning that they are always *urgent*, i.e., if at any state the output action is enabled, the system must execute that action;
- the input actions are never enabled unless the corresponding output actions from the environment synchronise with them: once they can be synchronised, they are urgent;
- for each mode, there is a self-loop labelled by the internal action.

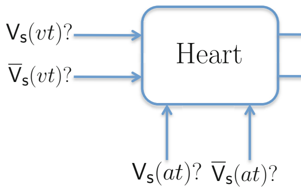
Definition 2 (Network of hybrid automata). *Let m be the number of HAs in the network. A state of the network is $\big((q^{(1)}, \eta^{(1)}), \cdots, (q^{(m)}, \eta^{(m)})\big)$. There is a transition*

$$\Big((q_i^{(1)}, \eta_i^{(1)}), \ldots, (q_i^{(m)}, \eta_i^{(m)})\Big) \rightarrow \Big((q_{i+1}^{(1)}, \eta_{i+1}^{(1)}), \ldots, (q_{i+1}^{(m)}, \eta_{i+1}^{(m)})\Big),$$

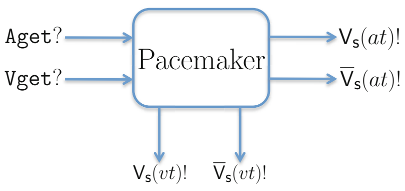
where

- *either, for each $1 \leq k \leq m$, $(q_i^{(k)}, \eta_i^{(k)})$ has a continuous evolution;*
- *or, for each $1 \leq k \leq m$, $(q_i^{(k)}, \eta_i^{(k)})$ has a discrete transition. If, for some k, $(q_i^{(k)}, \eta_i^{(k)})$ enables an output action $a \in E_2^{(k)}$, then all the other $(q_i^{(k')}, \eta_i^{(k')})$ must take a corresponding input action $a \in E_1^{(k')}$ (notice that this is guaranteed by input enabledness); otherwise, each state evolves by taking the internal action.*

We assume in our framework that both the heart model and the pacemaker model are specified as hybrid input-output automata. To allow user-specified models, we define fixed component interfaces for the heart and pacemaker models, as shown in Figure 1. The heart and the pacemaker communicate via input and output actions which are marked by ? and ! respectively. The pacemaker communicates with the heart through four output actions, $\mathsf{V_s}(at)!$, $\overline{\mathsf{V}}_\mathsf{s}(at)!$, $\mathsf{V_s}(vt)!$ and $\overline{\mathsf{V}}_\mathsf{s}(vt)!$. The actions $\mathsf{V_s}(at)!$ and $\overline{\mathsf{V}}_\mathsf{s}(at)!$ denote the beginning and the end of the *atrial* stimulus, respectively, while $\mathsf{V_s}(vt)!$ and $\overline{\mathsf{V}}_\mathsf{s}(vt)!$ denote the beginning and end of the *ventricle* stimulus. The heart communicates with the pacemaker using two output actions `Aget!` and `Vget!`.



(a) Heart model



(b) Pacemaker model

Fig. 1. Interfaces for the heart and pacemaker models

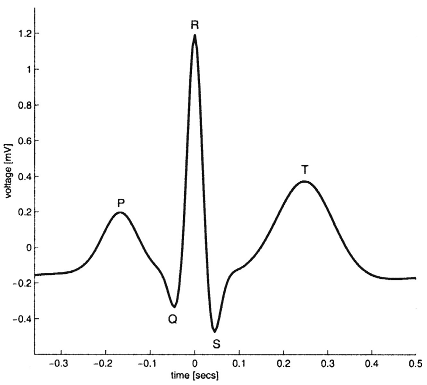
3 Heart Modelling

In this section we present two heart models, the ECG and the cardiac cell network model, and we show how they can be connected to the pacemaker and integrated within the overall verification framework. Each heart model has its own advantages and disadvantages. For instance, the ECG heart model can be easily adapted to a given patient, whereas the cardiac cell heart model is more physiologically relevant. By providing a common interface to the pacemaker, we can effectively evaluate and compare the behaviour of multiple heart models in a modular fashion.

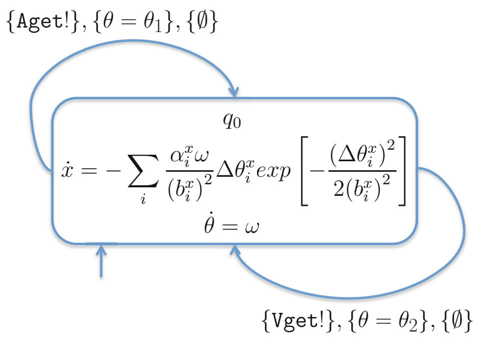
3.1 The ECG Heart Model

This heart model is based on synthetic ECG rhythms developed by Clifford *et al.* [8]. An ECG is a signal recorded from the surface of the human chest, which describes the activity of the heart. The ECG signal is an approximation of the electrical activity inside the human heart. An example ECG is given in Figure 2(a).

Typically, an ECG signal describes a cardiac cycle composed of three main waves, P, QRS and T. The P wave denotes the *atrial depolarisation*. The QRS wave reflects the rapid *depolarisation of the right and left ventricles*. The T wave denotes the *repolarisation of the ventricles*. In Figure 2(b) we present the hybrid automaton for the ECG heart model. It is based on a system on nonlinear ODE with two variables $x(t)$ (the value of the ECG signal at time t) and θ. Here θ_1 represents the beginning of the P wave; θ_2 represents the beginning of the Q wave; α_i^x and b_i^x, respectively, are the amplitude and width of the Gaussian functions used to model the ECG; $\theta \in [-\pi, \pi]$ is the *cardiac phase*; $\Delta\theta_i^x = (\theta - \theta_i^x) \bmod 2\pi$; and $\omega = \frac{2\pi h}{60\sqrt{h_{av}}}$ is the *angular velocity*, where h is the *instantaneous (beat-to-beat) heart rate* in BPM and h_{av} is the mean of the last n heart rates (typically with $n = 6$) normalized by 60 BPM. To use the ECG heart model one has to define the instantaneous (beat-to-beat) heart rate function $h(t)$ ($t \in \mathbb{R}_{\geq 0}$), which specifies the distance between two consecutive R-events (highest peak in Figure 2(a)). Technically, it is equivalent to the so called *RR-series* $\chi(n)$, $n \in \{1, \ldots, N\}$, where N denotes the length of the series.



(a) Example electrocardiogram [23]



$\{\texttt{Aget!}\}, \{\theta = \theta_1\}, \{\emptyset\}$
q_0
$\dot{x} = -\sum_i \frac{\alpha_i^x \omega}{(b_i^x)^2} \Delta\theta_i^x exp\left[-\frac{(\Delta\theta_i^x)^2}{2(b_i^x)^2}\right]$
$\dot{\theta} = \omega$
$\{\texttt{Vget!}\}, \{\theta = \theta_2\}, \{\emptyset\}$

(b) ECG hybrid automaton

Fig. 2. ECG heart model

The value of $\chi(n)$ denotes the time between two consecutive heart beats. More details on the construction of the function $h(t)$ can be found in [23].

3.2 The Cardiac Cell Heart Model

This heart model is based on modelling the *electrical conduction system (ECS)* of the heart (see [6]). The ECS is a network of nerves whose role is to propagate the *action potential (AP)* through the heart tissue. We abstract the conduction system as a network of cardiac cells, a model that is both physiologically meaningful and computationally tractable (in [6] we model a network of 33 cells). The ECS of the heart consists of conduction pathways with different *conduction delays*. Cells are connected by pathways. The delays of the pathways depend on the physiology of the tissue considered, and can be tuned to reproduce various tissue diseases.

Our model consists of the SA node, whose role is to generate sequences of AP signals which are propagated through the ECS of the heart, and 32 cells that share similar properties.

The cell model in Figure 3, taken from [26], consists of four modes, each associated with an AP phase: *resting and final repolarisation* (q_0), *stimulated* (q_1), *upstroke* (q_2), and *plateau and early repolarisation* (q_3). The cell model is characterised by two timed periods: effective refractory period (ERP) is the time period where the cell cannot be stimulated and relative refractory period (RRP) is the time period where a secondary excitation event is possible.

The variables of the model are: the membrane voltage v, which controls mode switches; i_{st}, which is the stimulus current; and a restitution-related variable v_n, used to modify the next ERP phase upon a new round of excitation. Specifically, this is achieved through the function $f(\lambda) = 1 + 13\sqrt[6]{\lambda}$ (mode q_3), where $\lambda = \frac{v_n}{V_R}$ and V_R is a model-specific constant called *repolarisation voltage* [26].

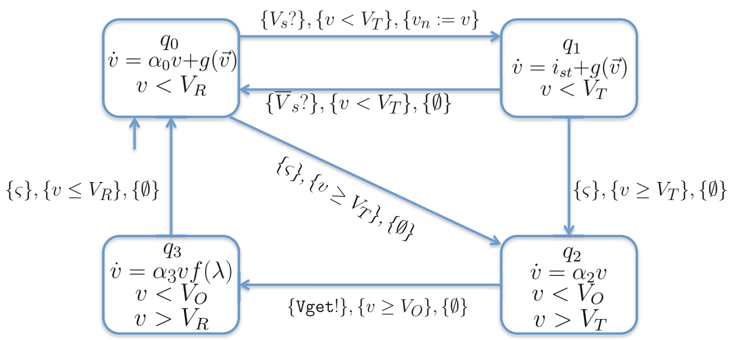

q_0
$\dot{v} = \alpha_0 v + g(\vec{v})$
$v < V_R$
$\{V_s?\}, \{v < V_T\}, \{v_n := v\}$
q_1
$\dot{v} = i_{st} + g(\vec{v})$
$v < V_T$
$\{\overline{V}_s?\}, \{v < V_T\}, \{\emptyset\}$
$\{\varsigma\}, \{v \leq V_R\}, \{\emptyset\}$
$\{\varsigma\}, \{v \geq V_T\}, \{\emptyset\}$
$\{\varsigma\}, \{v \geq V_T\}, \{\emptyset\}$
q_3
$\dot{v} = \alpha_3 v f(\lambda)$
$v < V_O$
$v > V_R$
$\{\texttt{Vget}!\}, \{v \geq V_O\}, \{\emptyset\}$
q_2
$\dot{v} = \alpha_2 v$
$v < V_O$
$v > V_T$

Fig. 3. Hybrid automaton for a ventricular cardiac cell

We denote with $\boldsymbol{v} = [v_1 \dots v_N]^T$ the vector of the membrane voltages of a network with N cells. We define a function $g_k(\boldsymbol{v})$ to express the voltage contribution to a cell k from the neighbouring cells, as follows:

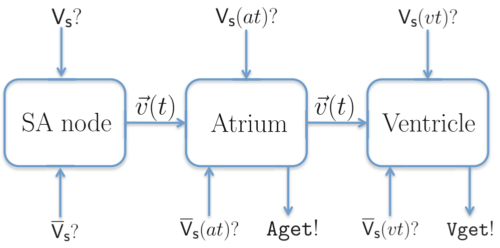
$$g_k(\boldsymbol{v}) = \sum_{i=1, i \neq k}^{N} v_i(t - \delta_{ki}) \cdot a_{ki} - v_k \cdot d_k, \tag{1}$$

where a_{ki} is the gain applied to the potential v_i from cell i, δ_{ki} is the time it takes for the potential to reach cell k, and d_k is the distance coefficient. These coefficients depend on the conduction system, and in particular on the conduction delays.

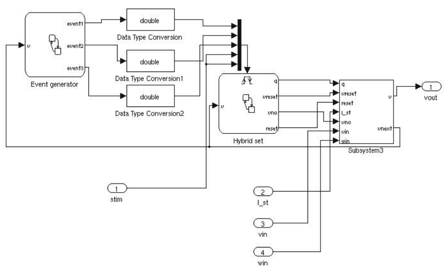
In Figure 4(a) we depict three blocks representing the connection of cells in the ECS. This component provides a template suitable for potentially including any kind of multi-cellular model of the cardiac tissue, and defines the interface with the pacemaker model.

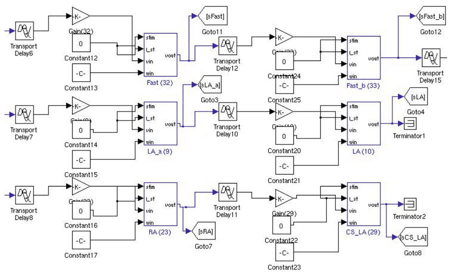
Every cell in the atrium and the ventricle blocks can be stimulated by the pacemaker using the input actions $\mathsf{V_s}(at)?$, $\overline{\mathsf{V}}_\mathsf{s}(at)?$ and $\mathsf{V_s}(vt)?$, $\overline{\mathsf{V}}_\mathsf{s}(vt)?$, respectively. The output actions `Aget!` and `Vget!` notify the pacemaker that the AP in the atrium and the ventricle (where the pacemaker leads are inserted) have reached a given threshold. The function $\boldsymbol{v}(t)$ is the output voltage from a given cell, which is the endpoint of the source block.

Figure 4(b) shows the Simulink implementation of a cardiac cell, which is given by three main blocks: *Event generator*, *Hybrid set* and *Subsystem*. The *Event generator* block is responsible for generating the input events to the cell. The *Hybrid set* implements the cell hybrid automaton model (see Fig. 3). The *Subsystem* block performs the integration procedure to compute the voltage level of the cell. In Figure 4(c), a simplified network of six cells is depicted. Each cell block is composed from the three sub-blocks shown in Figure 4(b) and connected to other cells through delay and gain components.


$\mathsf{V_s}$?
$\mathsf{V_s}(at)$?
$\mathsf{V_s}(vt)$?
SA node
$\vec{v}(t)$
Atrium
$\vec{v}(t)$
Ventricle
$\overline{\mathsf{V}}_\mathsf{s}$?
$\overline{\mathsf{V}}_\mathsf{s}(at)$?
Aget!
$\overline{\mathsf{V}}_\mathsf{s}(vt)$?
Vget!

(a) Electrical conduction system model





(b) Cell block (c) Cell connection

Fig. 4. Cardiac cell model

3.3 Switching between Different Heart Behaviours

The introduced heart models can exhibit only a single heart behaviour, such as normal, bradycardia or tachycardia, which is determined by the frequency of the RR-series (and sets the firing rate of the SA node in the cardiac cell model).

However, a real human heart exhibits several spontaneous changes of heart rhythms. In [6], we reproduce such dynamics by modelling the *probabilistic transition* between three modes, imposing a Normal (N), Bradycardia (B) and Tachycardia (T) rhythm, respectively, according to a prescribed RR-series for each mode. We also assume an initial distribution $\alpha \in \mathrm{Distr}(\{N, B, T\})$ and transition probabilities $\mathbf{P}_i \in \mathrm{Distr}(\{N, B, T\})$ for $i \in \{1, 2, 3\}$. We want to remark that both the initial distribution and the transition probabilities between behaviours can be learned from patient data, which enables the parametrization of personalized heart models.

4 Pacemaker Modelling

In this section we provide the specification of two pacemaker models in our framework. We consider the model by Jiang et al. [17], hereafter called the *basic pacemaker*, which is specified as a network of Timed Automata (TA); and an extension of the basic pacemaker presented in [6, 18], which we call the *enhanced pacemaker*, with advanced features such as sensing noise, energy consumption, and the ability to adapt the pacing rate depending on the physical activity of the patient.

4.1 Basic Pacemaker Model

The pacemaker is implanted under the chest skin and sends impulses to the heart at specific time intervals. The role of the basic pacemaker is to keep the heart rhythm at a given rate. It has two leads: one for the atrium and one for the ventricle. Each lead has the ability to sense or deliver an electrical signal.

The basic pacemaker model consists of five core TA components, named according to their specific function: LRI, AVI, URI, PVARP and VRP. The *lower rate interval (LRI)* component (Fig. 5(a)) has the function of keeping the heart rate above a given minimum value. The *atrio-ventricular interval (AVI)* component (Fig. 5(c)) is designed to maintain the synchronisation between the atrial and the ventricular events. An event is when the pacemaker senses or generates an action. The AVI component also defines the longest interval between an atrial event and a ventricular event. The *post ventricular atrial refractory period (PVARP)* component (Fig. 5(b)) notifies all other components that an atrial event has occurred. The *upper rate interval (URI)* component (Fig. 5(d)) sets a lower bound on the times between consecutive ventricular events. Finally, the *ventricular refractory period (VRP)* component (Fig. 5(d)) filters noise and early events that may cause undesired behaviour.

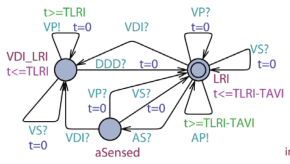
Three additional components, *Interval*, *Counter* and *Duration* (Figure 5(e) and (f)), are included in the basic pacemaker to detect and correct *pacemaker mediated tachycardia (PMT)*, an event occurring when the pacemaker increases the heart rate inappropriately. Such components switch the functioning modes of the pacemaker from DDD (pacing and sensing of the atrium and ventricle) to VDI (pacing and sensing only the ventricle). More details will be given in Section 5.1, and can be found in [6, 17].

There are four actions in the pacemaker model that serves as the interface with a generic heart model: the input actions Aget? and Vget? notify the pacemaker when there is an AP from the atrium or from the ventricle, respectively (see also Sect. 3.2), and likewise for the output actions AP! and VP! are responsible for pacing the atrium and the ventricle.

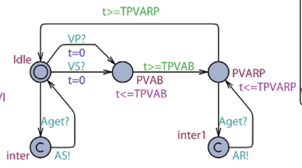
4.2 Enhanced Pacemaker Model

In this section, we extend the functionalities of the basic pacemaker model by considering noise, energy consumption and rate modulation through physiological sensors.

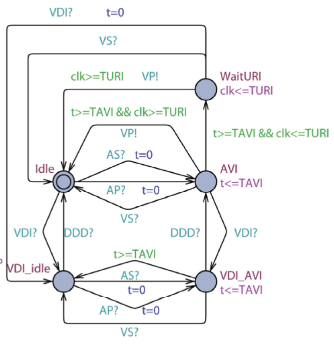
Pacing Noise. One of the important design issues of pacemakers is the need to tolerate noise. For instance, when the pacemaker tries to deliver a beat, the beat might get lost due to noise on the channel. The basic pacemaker is constructed under the simplified assumption that it can pace the heart perfectly. Here we



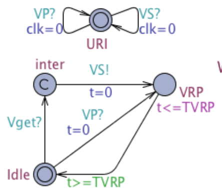
(a) LRI component



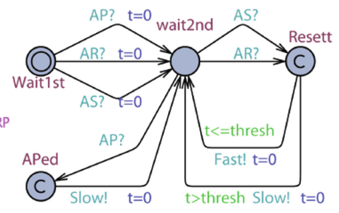
(b) PVARP component



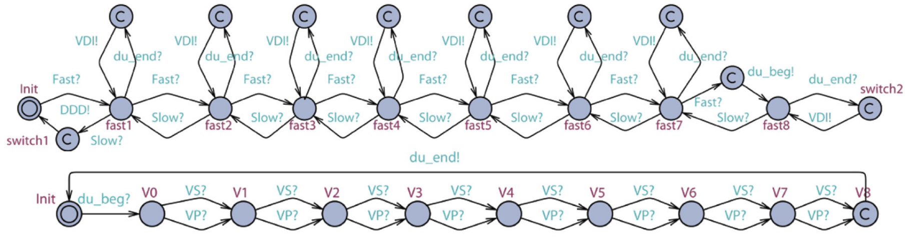
(c) AVI component



(d) URI and VRP components



(e) Interval component



(f) Counter and Duration component

Fig. 5. Timed automata of the five core pacemaker components (a,b,c,d), and of Interval, Counter and Duration components for PMT analysis (e,f). Locations labelled with **C** indicate committed locations that do not allow time to elapse.

consider a more realistic scenario, modelling the so called "failure-to-capture", a kind of sensing noise due to insufficient contact between the lead and the myocardium, or due to lead fracture [11]. In particular, we add to the fixed stimulus current i_{st} (cf. Figure 3) a *normally distributed noise* with mean μ and variance σ^2 each time the pacemaker wants to pace the cell.

In this way, if the noise added to the channel is too high, a "missing stimulus" is generated, i.e. the stimulus from the pacemaker will *not* be high enough to stimulate the cell.

Energy. Pacemaker's life time is limited and is crucially dependent on the battery embedded into the devices. When the battery depletes, the pacemaker needs to be re-implanted, and hence the analysis of energy usage and, ultimately, the design of more energy-efficient devices are indispensable.

In our framework, we consider the so called *Kinetic Battery model (KiBaM)* [22] to describe the dynamics of energy consumption. The model consists of the following system of ODEs:

$$\frac{dy_1(t)}{dt} = -\iota(t) + k\left(\frac{y_2(t)}{1-c} - \frac{y_1(t)}{c}\right), \quad \frac{dy_2(t)}{dt} = -k\left(\frac{y_2(t)}{1-c} - \frac{y_1(t)}{c}\right). \quad (2)$$

The battery charge is distributed in two wells: the *available-charge* $y_1(t)$ and the *bound-charge* $y_2(t)$. The current applied to the battery at time t is described by the function $\iota(t)$. When the value of $\iota(t)$ is zero the battery enters the recovery mode, where the energy from the bound-charge well flows to the available-charge well. This mode allows a nearly discharged battery to recover in a period of zero or low current by increasing its available-charge. When the current $\iota(t)$ is not zero, both charges $y_1(t)$ and $y_2(t)$ decay over time. The battery is considered to be empty when there is no charge in the available-charge well, i.e., $y_1(t) = 0$. For details on the composition between the KiBaM and the pacemaker model see [6].

Rate Adaptive Pacemaker. Physiological sensors are an essential component of the so-called *rate adaptive (RA) pacemaker*, where the pacing rate is adjusted according to the levels of activity (physical, mental or emotional) detected in the patient. RA pacemakers represent the only choice for individuals with chronotropic incompetence, that is, when the heart rate cannot naturally adapt to increasing demand (e.g. AV block). A number of different pacing methods and sensors have been developed so far [2]. However, they require extensive testing on cardiac patients especially to assess the device under varying levels of physical exercise. Our model-based framework provides an effective test-bed for these kinds of devices, where different (and possibly multiple) sensors can be integrated into available pacemaker models, and formal verification enables the automated design and debugging of rate modulation protocols in order to ensure safe behaviour of the heart under the different stress levels which the patient can undergo.

In [18], we develop a HIOA model of a *VVIR pacemaker* (sensing and pacing of the ventricle, and with rate modulation) based on a *QT interval (QTI) sensor*, a highly specific metabolic sensor that exploits the fact that physical activity shortens the QT interval (see Fig. 2a), and thus requires an increased heart rate. We implement the QT sensor through a runtime ECG detection algorithm that allows to simulate and validate the model with patient ECG data.

The *RA component* (Fig. 6) is connected to the components of the VVI pacemaker and is responsible for changing the pacing rate (TLRI) according to the signals from the QT sensor, which outputs an action $\mathsf{TE}!$ whenever a T wave is detected. To this aim, we established a relationship between QTI lengths and

TLRI by means of a non-linear regression analysis performed over ECG data from the PhysioNet database [1], and described by the following equation

$$\mathsf{RR}(\mathsf{QT}) = -\frac{\log((a - \mathsf{QT})/b)}{k} \tag{3}$$

where a, b and k are the estimated regression parameters; QT is the QTI length; and RR is the RR interval length which is used to update TLRI.

From the initial state q_0, the RA component waits for a ventricle sense or pace event (Vget or VP, resp.) to start timers t_{VP} and t_{QT}. t_{VP} defines the refractory window of size T_R where the RA component disables the pacemaker inputs, while t_{QT} models the duration of the QT interval and is terminated by the synchronization with a $\mathsf{TE!}$ signals. If the obtained t_{QT} falls within an admissible interval $[T^l, T^u]$, the corresponding adapted value for TLRI is calculated through function f_{QT}, which applies the above regression law over the mean of the last four detections. This averaging mechanism ensures prompt response to fast changing QTIs and, at the same time, allows us to mitigate the effects of wrongly sensed intervals.

The ECG detection algorithm implemented in the QT sensor component relies on a signal processing algorithm based on [12, 27], and is thoroughly explained in [18]. Note that the behaviour of the QT sensor is inherently stochastic, since it processes and filters ECG signals that are subject to random noise. However, other sources of uncertainty can be incorporated, like random under- and over-sensing.

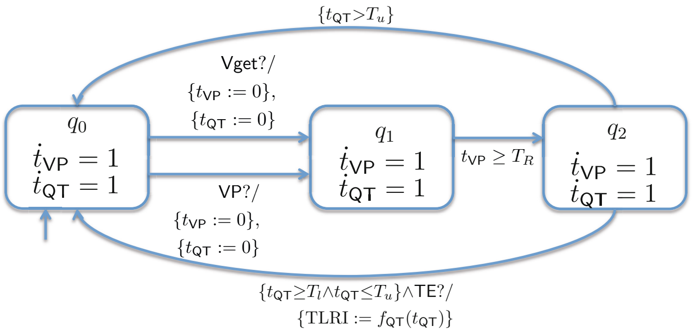

$\{t_{\mathsf{QT}}>T_u\}$
$\mathsf{Vget}?/$ $\{t_{\mathsf{VP}}:=0\}$, $\{t_{\mathsf{QT}}:=0\}$
q_0 $\dot{t}_{\mathsf{VP}}=1$ $\dot{t}_{\mathsf{QT}}=1$
q_1 $\dot{t}_{\mathsf{VP}}=1$ $\dot{t}_{\mathsf{QT}}=1$
$t_{\mathsf{VP}}\geq T_R$
q_2 $\dot{t}_{\mathsf{VP}}=1$ $\dot{t}_{\mathsf{QT}}=1$
$\mathsf{VP}?/$ $\{t_{\mathsf{VP}}:=0\}$, $\{t_{\mathsf{QT}}:=0\}$
$\{t_{\mathsf{QT}}\geq T_l \wedge t_{\mathsf{QT}}\leq T_u\}\wedge\mathsf{TE}?/$
$\{\mathrm{TLRI}:=f_{\mathsf{QT}}(t_{\mathsf{QT}})\}$

Fig. 6. Hybrid automaton of the rate adaptive component

5 Pacemaker Verification

In this section, we report some experimental results obtained in the evaluation of our framework with the basic and the enhanced pacemaker models. All the following experiments have been performed with the cardiac cell model. First, we show how the pacemaker corrects bradycardia when the probability of deviating from the normal behaviour is varied, and how cases of pacemaker mediated tachycardia are solved by mode switching. Second, we evaluate the behaviour of our model under different levels of sensing noise; we analyse the energy consumption of the pacemaker and its dependence on the pacing rate; and we conduct experiments for the rate-modulation property, considering multiple inputs from the QT sensor and physical exercise curves.

5.1 Verification of the Basic Pacemaker Model

Probabilistic Switching. We conduct experiments considering the probabilistic transitions between different heart behaviours, as explained in Sect. 3.3.

In this analysis, we obtain a relationship between the probability to generate bradycardia and the number of pacemaker beats to the ventricle, shown in Figure 7. We range the probability from 0.05 to 0.95 and run 40 experiments, each representing 8 minutes of heart beat. We clearly observe that, by increasing the probability of a bradycardia behaviour, the pacemaker delivers more beats to the ventricle. This gives evidence for the ability of our pacemaker to correct random bradycardia episodes.

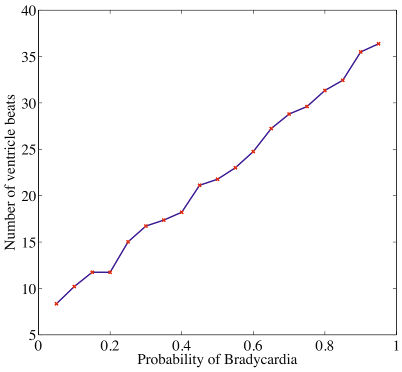


Fig. 7. Paced ventricular beats at varying probabilities of Bradycardia behaviour

Pacemaker Mediated Tachycardia. In human hearts, the atrium can beat faster than the ventricle, at ratio 2:1 or 3:1. The resulting heart beat can still be regular due to a special cell called the AV node, which has a blocking period longer than the other cells. The AV node connects the ECS of the atrium to the ECS of the ventricle. The pacemaker tries to maintain a 1:1 AV conduction through the AVI component. Thus, in the event of PMT, the pacemaker increases the beats in the ventricle inappropriately. In order to avoid this behaviour we need to switch the pacemaker from the DDD mode to the VDI mode when the PMT event is detected. After PMT is successfully corrected and a normal heart beat is re-established, the pacemaker can switch back to the DDD mode.

In Figure 8 we show an experiment where a tachycardia episode in the ventricle due to PMT (red curve), is corrected by a mode switch from DDD to VDI at time 13. As a result, the number of ventricle beats decreases and the regular heart rhythm is recovered (blue curve).

5.2 Verification of the Enhanced Pacemaker Model

Noise. Here we address the occurrence of random "failure to capture" events, generated by the presence of random noise on the pacemaker leads (illustrated

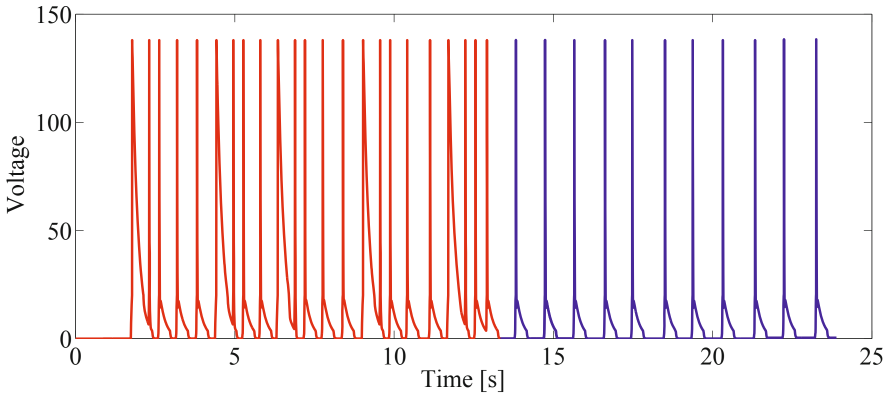


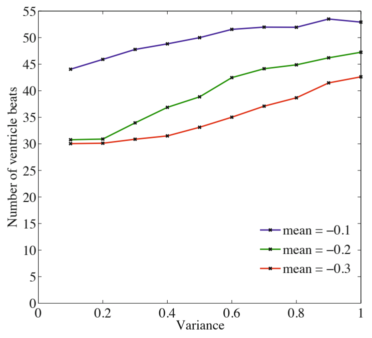
Fig. 8. AP in the ventricle during a PMT episode. The red curve shows a tachycardia frequency, corrected through mode switch at time 13 (blue curve).

in Section 4.2). In the following experiments, two parameters are considered: the mean μ and the variance σ^2 of the normally distributed noise. Figure 9(a) shows the number of ventricular beats for different values of μ (red line with $\mu = -0.3$, green line with $\mu = -0.2$ and blue line with $\mu = -0.1$). We choose a negative μ in order to simulate the undersensing effect. In each experiment with fixed mean μ, we make the variance range from 0.1 to 1 with step of 0.1.

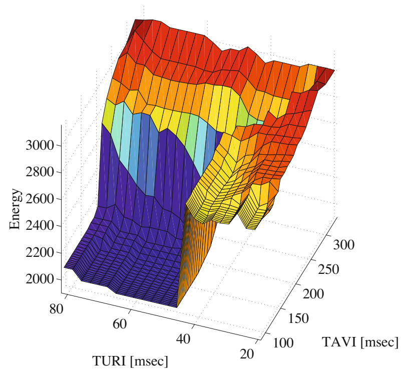
The results demonstrate that, when Gaussian noise with small mean (indicating a high degree of undersensing) is added to the stimulus, the number of beats in the ventricle decreases, since more beats induced by the pacemaker will be lost. On the other hand, increasing the variance of the normal distribution will yield a higher number of beats. Indeed, higher variance to the noise, when centred at negative mean, produces better chances of picking positive samples from the normal distribution. This, in turn, implies a better chance for the stimulus to be high enough to stimulate the cell.

Energy. In this analysis, we are interested in the energy consumption of the pacemaker when setting the SA node to induce bradycardia, thus forcing the device to deliver paced beats. Figure 9(b) shows the results obtained by varying two parameters, TAVI and TURI, which are the default programmable parameters used by technicians to ensure a heart beat between 60 and 100 BPM. We make TAVI range in the interval $[70 - 300]$ ms with 10 ms increment, and the value of TURI in $[50 - 175]$ BPM with 5 BPM increment. Fig. 9(b) evidences a steep increase in energy consumption when TURI$<$ 50 or TAVI$>$ 200. This behaviour is caused by the fact that we are forcing the pacemaker to wait less between two consecutive ventricular events. Therefore, the pacemaker will initiate most of the ventricular beats before the occurrence of a natural beat, thus leading to a more prominent depletion of battery charge.

Parametric Analysis and Sensor Induced Tachycardia. We perform an exhaustive parameter exploration for evaluating the behaviour of the rate



(a) Number of ventricle beats with random undersensing at different variances.



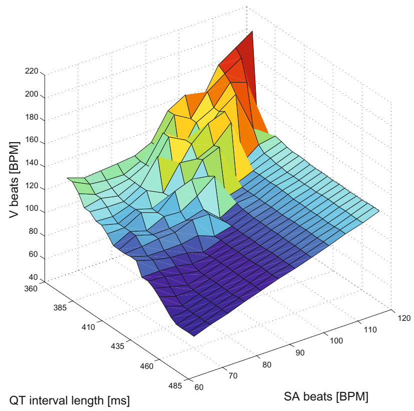
(b) Battery charge in 1 min period under Bradycardia, at varying TAVI and TURI.

Fig. 9. Sensing noise (a) and energy consumption (b) experiments in the enhanced pacemaker verification

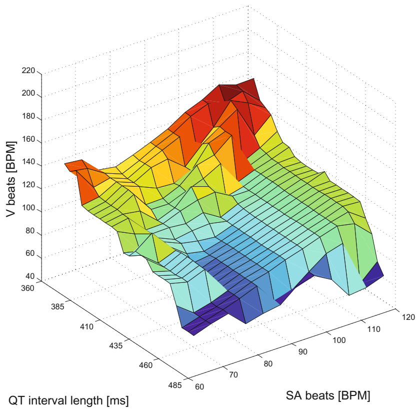
adaptive pacemaker model over a wide spectrum of firing rates of the sinus node (SA node) and QTI lengths. The SA frequency models the ideal heart rate demand and expresses the levels of stress and activity; the QTI lengths detected by the QT sensor are used to update the pacing rate as illustrated in Sect. 4.2. For evident vital reasons, the application on real devices and patients of this kind of quantitative analyses can involve only a limited range of safe parameter settings and feasible activity levels, and is therefore insufficient for assessing the effects of sensors faults and of extreme SA rates.

Instead, with our formal framework, we can distinguish the parameter regions under which the pacemaker correctly operates from those where phenomena of *sensor-induced tachycardia (SIT)* occur, i.e. when sensors malfunctioning (in our case, wrongly detected short QTIs) lead to inappropriately fast pacing rate. Figure 10 compares the number of ventricular beats in healthy conditions (a) and in presence of AV block (b), over 552 different combinations of QTI lengths and SA firing rates.

Such analysis provides evidence of a diagonal threshold of ideal QTI lengths and SA rates, below which we observe a SIT phenomenon, characterized by a ventricular rate constantly higher than the SA rate, which is amplified as the QTI decreases. This faulty behaviour is slightly less evident in the AV block scenario, because of the number of beats lost by the defective AV node. On the other hand, if for each SA rate appropriate QTIs are considered (above the ideal threshold), we observe a regular pattern in the number of ventricular beats. With a healthy AV node, they increase linearly in the number of SA beats, thus reproducing a correct conduction system. In the case of AV block, the frequency in the ventricle grows linearly before reaching a final plateau, indicating the inability to deliver high frequencies.



(a) V beats, Normal AV node



(b) V beats, AV block

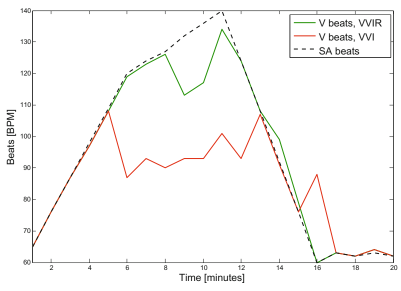
Fig. 10. Number of ventricular beats (z-axis) over multiple QTIs (x-axis) and SA node firing frequencies (y-axis)

Modulation during Physical Activity. We validate our VVIR model by comparing it to its fixed-rate counterpart (VVI) over typical exercise curves of a young (Fig. 11(a)) and old (Fig. 11(b)) individual. Heart rate during physical exercise is characterized by four stages: neural slope (initial fast increase); metabolic slope (slower increase); decay (fast decrease during recovery); and resting. Since old subjects cannot generally provide the same exercise intensity as young individuals, their activity curves are characterized by a lower maximum heart rate.

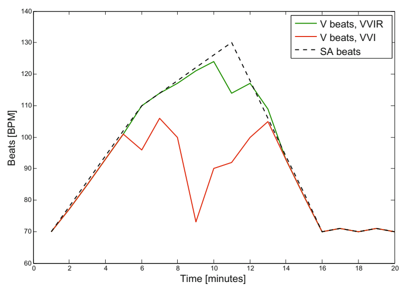
Results during a 20 minutes exercise demonstrate that our VVIR implementation successfully manages to modulate the pacing rate according to the intensity of physical activity in both classes of patients. Minor rate discrepancies occur in the most intense phases; these are, however, negligible if compared to the behaviour of the fixed rate pacemaker (unable to provide an appropriate rate at SA rates higher than 110 BPM). Moreover, no SIT events are detected during exercise, regardless the intensity of physical activity.

6 Future Directions

In the previous section we described the verification of the pacemaker model together with two heart models: the ECG and the cardiac cell network. In future, we plan to use more advanced heart models to capture the physiological characteristics of the heart. Also, we plan to synthesise crucial timing parameters of the pacemaker model such that it satisfies a given specification.



(a) Young patient



(b) Old patient

Fig. 11. Rate modulation during exercise in young (a) and old (b) patients. The SA rate (black dashed line) gives the metabolic demand following typical activity curves. The number of ventricular beats is compared between the VVIR (green curve) and the VVI (red curve) pacemakers.

6.1 The Minimal Ventricular Cardiac Cell Heart Model

As an alternative to the previous heart models we propose to use the minimal ventricular (MV) model of Bueno-Orovio et al. [3]. Unlike the previous two models, the MV model can reproduce realistic and important AP phenomena, e.g. alternans [14], and yet is computationally more efficient than some of the other models in the literature. Using the techniques from Grosu et al. [13], we can abstract the MV model into a network of hybrid automata (see Figure 12) that fits our developed framework for pacemaker verification. For details see [15].

The MV model describes the flow of currents through a cell. The model is defined by four nonlinear PDEs representing the transmembrane potential $x_1(\boldsymbol{d}, t)$, the fast channel gate $x_2(\boldsymbol{d}, t)$, and two slow channel gates, $x_3(\boldsymbol{d}, t)$ and $x_4(\boldsymbol{d}, t)$. All of the four variables are time and position $\boldsymbol{d} := (d_x, d_y, d_z) \in \mathbb{R}^3$ dependent. For one dimensional tissue, i.e., $\boldsymbol{d} := d_x$, the evolution of transmembrane potential is given by:

$$\frac{\partial x_1(d_x,t)}{\partial t} = D\frac{\partial^2 x_1(d_x,t)}{\partial d_x^2} + e(x_1,t) - (J_{\text{fi}} + J_{\text{so}} + J_{\text{si}}), \tag{4}$$

where $D \in \mathbb{R}$ is the diffusion coefficient, $e(\boldsymbol{d}, t)$ is the external stimulus applied to the cell, J_{fi} is the fast inward current, J_{si} is the slow inward current and J_{so} is the slow outward current. The currents J_{fi}, J_{so} and J_{si} are described by Heaviside function. For more details see [3]. To define the propagation of the action potential on a cardiac ring of length L, we set the boundary conditions to: $x_i(0, t) = x_i(L, t)$ for all $i \in \{0, \ldots, 4\}$ and $t \in \mathbb{R}$.

HA Approximation. One alternative to solving these highly nonlinear PDEs is to discretize space and hybridize the dynamics. The result is the HA model. Following the approach of [13], we first hybridize the dynamics and obtain a HA

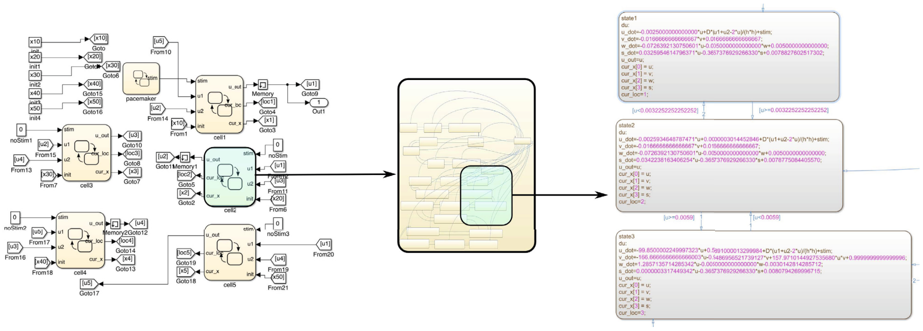


Fig. 12. Left: top-level Simulink/Stateflow model for a ring of five cardiac cells; the *Pacemaker* block stimulates one cell. Center: Stateflow model of a single cardiac cell. Right: dynamics and guards in 3 locations of a single cell.

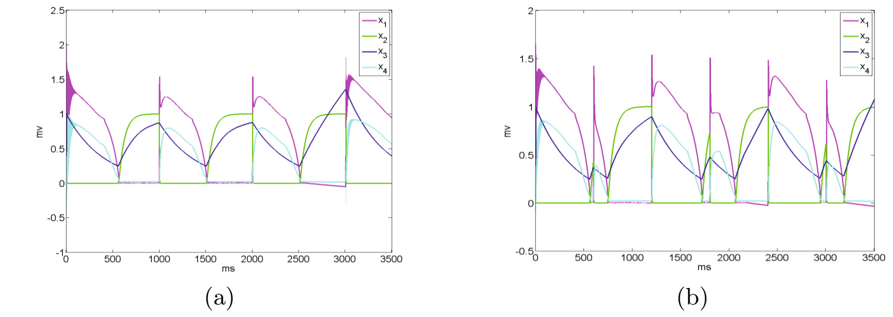


Fig. 13. Reach set projected on x_{11} (AP) for stimulation period of 1000 msec (a) and 600 msec (b) with x-axis for time and y-axis for voltage

with 29 locations. The basic idea is to approximate the Heaviside function from J_{fi}, J_{so} and J_{si} with a sequence of ramp functions. Each location of the resulting HA contains a multi-affine ODE such as:

$$\dot{x_1} = -0.935x_1 + 12.70x_2 - 8.0193x_1x_2 + 0.529x_3x_4 + 0.87 + st$$
$$\dot{x_2} = -0.689x_2;\ \dot{x_3} = -0.0025x_3;\ \dot{x_4} = 0.0293x_1 - 0.0625x_4 + 0.0142,$$

where st is the time-varying stimulus input. The 29 locations represent the final HA model of a single cardiac cell. By discretising the spatial location we obtain a network of cells that can be connected in a ring or in a tree depending on the physiological characteristics of the heart that we would like to model. In Figure 12 we depict a Simulink/Stateflow implementation of 5 cardiac cells connected in a ring; in Figure 13 we depict the voltage level of a cardiac cell for a set of initial conditions.

In [15] we have developed techniques on how to compute the over-approximation of the reach set, i.e., the voltage level of the cardiac cell at a given time moment, for a network of cardiac cells given by the MV model. As a future direction we plan to connect the minimal ventricular cardiac cell heart model to the pacemaker model, and investigate more advanced specifications such as linear duration properties [7].

6.2 Automated Synthesis of Pacemaker Software

Pacemaker devices have a limited number of programmable parameters and there is considerable agreement among manufacturers on the appropriate values to set according to the considered heart condition, implying that the same pacemaker settings are used for large classes of cardiac patients exhibiting the same disease.

We believe that model synthesis methods can significantly advance the automated design of *highly personalized* pacemaker devices where, instead of choosing among a limited number of condition-specific settings, parameters are automatically derived according to patient's clinical history, and continuously adapted to reflect the real-time monitoring of her/his conditions.

Given that patient-specific models of the heart can be constructed from the detailed electro-physiological data obtainable with current diagnostic means, and given a (formal) property describing the desired behaviour of the heart, synthesis techniques would provide pacemaker models that are correct-by-design, so that, when composed with the heart model, the required behaviour is ensured without the burden of formally verifying the property against all the possible combinations of parameters.

Moreover, synthesis methods for implantable pacemakers need to cope with a range of *uncontrollable parameters*, which, unlike the controllable timings of a pacemaker, cannot be adjusted to our needs. Think, for example, about the timing at which ventricle beats are fired, or any other physiological parameter of the heart. Hence, the purpose is to find a positive solution to an *optimal synthesis problem*, which consists in finding optimal values for the controllable parameters such that the composed heart-pacemaker model meets the required (healthy) behaviour specification, regardless of the uncontrollable parameters. As usual, the notion of optimality underlies a quantitative objective function we want to maximize or minimize (e.g. energy consumption).

In [9], an initial approach to the optimal synthesis of pacemaker devices is proposed, based on symbolic constraint reasoning, and with application to networks of timed I/O automata. We are currently working to extend the approach towards richer and more complex models, featuring hybrid and probabilistic dynamics.

7 Conclusion

In this paper, we presented a model-based framework for the formal analysis of implantable pacemakers, which supports the plug-in and the integration of different heart and pacemaker models by means of a small set of pre-defined interfaces and modelling templates. In the composed heart-pacemaker model, stochasticity comes into play in several ways, such as in the probabilistic behaviour of the heart or in the occurrence of random sensor faults. The framework enables the analysis of a broad range of electro-physiological and device-related properties, computed through simulation or quantitative verification, thus providing safety guarantees of the pacemaker in presence of multiple sources of uncertainty. Tool support is of crucial importance, and we, indeed, provide a sound implementation of the formal framework in MATLAB Simulink/Stateflow.

We evaluated our framework over two heart models, the ECG and the cardiac cell model; and over an enhanced pacemaker design, built by modularly adding advanced functionalities, including energy, sensor noise and rate-adaptation on top of a basic model inspired by [17]. We reported here only some of the experimental results obtained in previous work [4–6, 18], showing how quantitative verification can provide practical guidance for safer and more efficient designs of pacemaker devices, and, ultimately, give insight into the defective dynamics of heart diseases.

Current research efforts are directed towards the analysis of more advanced and physiologically accurate heart models, and to the synthesis of pacemaker parameters. As future work, we aim to formulate and implement novel synthesis methods, able to automatically derive not just pacemaker parameters, but also complete specifications of pacing algorithms and protocols, which are optimal under safety and cost-effectiveness, and account for the stochastic dynamics of the heart and sensors.

Acknowledgments. This work is supported by the ERC AdG VERIWARE, ERC PoC VERIPACE and the Institute for the Future of Computing, Oxford Martin School.

References

1. PhysioNet, `http://www.physionet.org/physiobank/`
2. Barold, S.S., Stroobandt, R.X., Sinnaeve, A.F.: Cardiac pacemakers and resynchronization step by step: An illustrated guide. John Wiley & Sons (2010)
3. Bueno-Orovio, A., Cherry, E.M., Fenton, F.H.: Minimal model for human ventricular action potentials in tissue. Journal of Theoretical Biology 253(3), 544–560 (2008)
4. Chen, T., Diciolla, M., Kwiatkowska, M., Mereacre, A.: Quantitative verification of implantable cardiac pacemakers. In: 2012 IEEE 33rd Real-Time Systems Symposium (RTSS), pp. 263–272. IEEE (2012)
5. Chen, T., Diciolla, M., Kwiatkowska, M., Mereacre, A.: A simulink hybrid heart model for quantitative verification of cardiac pacemakers. In: Proceedings of the 16th International Conference on Hybrid Systems: Computation and Control (HSCC 2013), pp. 131–136 (2013)
6. Chen, T., Diciolla, M., Kwiatkowska, M., Mereacre, A.: Quantitative verification of implantable cardiac pacemakers over hybrid heart models. Information and Computation (in press, 2014)
7. Chen, T., Diciolla, M., Kwiatkowska, M.Z., Mereacre, A.: Verification of linear duration properties over continuous-time markov chains. ACM Trans. Comput. Log. 14(4), 33 (2013)
8. Clifford, G., Nemati, S., Sameni, R.: An Artificial Vector Model for Generating Abnormal Electrocardiographic Rhythms. Physiological Measurements 31(5), 595–609 (2010)
9. Diciolla, M.: Quantitative Verification of Real-Time Properties with Application to Medical Devices. PhD thesis, Department of Computer Science. University of Oxford (2014)

10. Gomes, A.O., Oliveira, M.V.M.: Formal specification of a cardiac pacing system. In: Cavalcanti, A., Dams, D.R. (eds.) FM 2009. LNCS, vol. 5850, pp. 692–707. Springer, Heidelberg (2009)
11. Greenhut, S., Jenkins, J., MacDonald, R.: A stochastic network model of the interaction between cardiac rhythm and artificial pacemaker. IEEE Transactions on Biomedical Engineering 40(9), 845–858 (1993)
12. Gritzali, F., Frangakis, G., Papakonstantinou, G.: Detection of the P and T waves in an ECG. Computers and Biomedical Research 22(1), 83–91 (1989)
13. Grosu, R., Batt, G., Fenton, F.H., Glimm, J., Le Guernic, C., Smolka, S.A., Bartocci, E.: From cardiac cells to genetic regulatory networks. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 396–411. Springer, Heidelberg (2011)
14. Guevara, M.R., Ward, G., Shrier, A., Glass, L.: Electrical alternans and period-doubling bifurcations. Computers in Cardiology, 167–170 (1984)
15. Huang, Z., Fan, C., Mereacre, A., Mitra, S., Kwiatkowska, M.: Invariant verification of nonlinear hybrid automata networks of cardiac cells. In: Biere, A., Bloem, R. (eds.) CAV 2014. LNCS, vol. 8559, pp. 373–390. Springer, Heidelberg (2014)
16. Jiang, Z., Pajic, M., Connolly, A., Dixit, S., Mangharam, R.: Real-time heart model for implantable cardiac device validation and verification. In: ECRTS, pp. 239–248 (2010)
17. Jiang, Z., Pajic, M., Moarref, S., Alur, R., Mangharam, R.: Modeling and verification of a dual chamber implantable pacemaker. In: Flanagan, C., König, B. (eds.) TACAS 2012. LNCS, vol. 7214, pp. 188–203. Springer, Heidelberg (2012)
18. Kwiatkowska, M., Lea-Banks, H., Mereacre, A., Paoletti, N.: Formal modelling and validation of rate-adaptive pacemakers. In: IEEE International Conference on Healthcare Informatics 2014, ICHI 2014 (to appear, 2014)
19. Lian, J., Krätschmer, H., Müssig, D., Stotts, L.: Open source modeling of heart rhythm and cardiac pacing. Open Pacing Electrophysiol. Ther. J. 3, 4 (2010)
20. Lynch, N., Segala, R., Vaandrager, F., Weinberg, H.B.: Hybrid I/O automata. In: Alur, R., Sontag, E.D., Henzinger, T.A. (eds.) HS 1995. LNCS, vol. 1066, pp. 496–510. Springer, Heidelberg (1996)
21. Macedo, H.D., Larsen, P.G., Fitzgerald, J.: Incremental Development of a Distributed Real-Time Model of a Cardiac Pacing System Using VDM. In: Cuellar, J., Sere, K. (eds.) FM 2008. LNCS, vol. 5014, pp. 181–197. Springer, Heidelberg (2008)
22. Manwell, J.F., McGowan, J.G.: Lead acid battery storage model for hybrid energy systems. Solar Energy 50(5), 399–405 (1993)
23. McSharry, P.E., Clifford, G.D., Tarassenko, L., Smith, L.A.: A dynamical model for generating synthetic electrocardiogram signals. IEEE Transactions on Biomedical Engineering 50(3), 289–294 (2003)
24. Méry, D., Singh, N.K.: Pacemaker's Functional Behaviors in Event-B. Rapport de recherche, MOSEL - INRIA Lorraine - LORIA (2009)
25. Tuan, L.A., Zheng, M.C., Tho, Q.T.: Modeling and verification of safety critical systems: A case study on pacemaker. In: 2010 Fourth International Conference on Secure Software Integration and Reliability Improvement (SSIRI), pp. 23–32. IEEE (2010)
26. Ye, P., Entcheva, E., Grosu, R., Smolka, S.A.: Efficient modeling of excitable cells using hybrid automata. In: Proc. of CMSB, vol. 5, pp. 216–227 (2005)
27. Yeh, Y.C., Wang, W.J.: QRS complexes detection for ECG signal: The Difference Operation Method. Computer Methods and Programs in Biomedicine 91(3), 245–254 (2008)

Model Checking Hybrid Systems
(Invited Talk)

Edmund M. Clarke and Sicun Gao

Carnegie Mellon University

Abstract. We present the framework of delta-complete analysis for bounded reachability problems of hybrid systems. We perform bounded reachability checking through solving delta-decision problems over the reals. The techniques take into account of robustness properties of the systems under numerical perturbations. Our implementation of the techniques scales well on several highly nonlinear hybrid system models that arise in biomedical applications.

1 Introduction

Formal verification is difficult for hybrid systems with nonlinear dynamics and complex discrete controls [1,7]. A major difficulty of applying advanced verification techniques in this domain comes from the need of solving logic formulas over the real numbers with nonlinear functions, which is notoriously hard.

Recently, we have defined the *δ-decision problem* that is much easier to solve [3,2]. Given an arbitrary positive rational number δ, the δ-decision problem asks if a logic formula is false or *δ-true* (or, dually, true or *δ-false*). The latter answer can be given, if the formula *would be true* under δ-bounded numerical changes on its syntactic form [3]. The δ-decision problem is decidable for bounded first-order sentences over the real numbers with arbitrary Type 2 computable functions. Type 2 computable functions [8] are essentially real functions that can be approximated numerically. They cover almost all functions that can occur in realistic hybrid systems, such as polynomials, trigonometric functions, and solutions of Lipschitz-continuous ODEs. We can now develop a new framework for solving bounded reachability problems for hybrid systems based on solving δ-decisions. We show that this framework makes bounded reachability of hybrid systems much more tractable. Moreoever, our practical implementation can handle highly nonlinear hybrid systems.

The framework of *δ-complete analysis* consists of techniques that perform verification and allow bounded errors on the safe side. For bounded reachability problems, δ-complete analysis aims to find one of the following answers:

- safe (bounded): The system does not violate the safety property within a bounded period of time and a bounded number of discrete mode changes.
- δ-unsafe: The system would violate the safety property under some δ-bounded numerical perturbations.

Thus, when the answer is safe, no error is involved. On the other hand, a system that is δ-unsafe would violate the safety property under bounded numerical perturbations. Realistic hybrid systems interact with the physical world and it is impossible to avoid slight perturbations. Thus, δ-unsafe systems should indeed be regarded as unsafe, under reasonable choices of δ. Note that such robustness problems can not be discovered by solving the precise decision problem, and the use of δ-decisions strengthens the verification results.

δ-Complete reachability analysis reduces verification problems to δ-decision problems of formulas over the reals. It follows from δ-decidability of these formulas [3] that δ-complete reachability analysis of a wide range of nonlinear hybrid systems is decidable. Such results stand in sharp contrast to the standard high undecidability of bounded reachability for simple hybrid systems.

We emphasize that the new framework is immediately practical. We implemented the techniques in our open-source tool dReach based on our nonlinear SMT solver dReal [4]. In our previous work, we have shown the underlying solver scales on nonlinear systems [5]. The tool has successfully verified safety properties of various nonlinear models that are beyond the scope of existing tools, such as the cardiac cells model as studied in [6].

References

1. Alur, R.: Formal verification of hybrid systems. In: EMSOFT, pp. 273–278 (2011)
2. Gao, S., Avigad, J., Clarke, E.M.: Delta-complete decision procedures for satisfiability over the reals. In: Proceedings of the 6th International Joint Conference on Automated Reasoning (IJCAR), pp. 286–300 (2012)
3. Gao, S., Avigad, J., Clarke, E.M.: Delta-decidability over the reals. In: Proceedings of the 27th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), pp. 305–314 (2012)
4. Gao, S., Kong, S., Clarke, E.M.: dReal: An SMT solver for nonlinear theories over the reals. In: Bonacina, M.P. (ed.) CADE 2013. LNCS (LNAI), vol. 7898, pp. 208–214. Springer, Heidelberg (2013)
5. Gao, S., Kong, S., Clarke, E.M.: Satisfiability modulo ODEs. In: Proceedings of the 13th International Conference on Formal Methods in Computer Aided Design, FMCAD (2013)
6. Grosu, R., Batt, G., Fenton, F.H., Glimm, J., Le Guernic, C., Smolka, S.A., Bartocci, E.: From cardiac cells to genetic regulatory networks. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 396–411. Springer, Heidelberg (2011)
7. Henzinger, T.A.: The theory of hybrid automata. In: LICS, pp. 278–292 (1996)
8. Weihrauch, K.: Computable Analysis: An Introduction (2000)

Challenges for the Dynamic Interconnection of Medical Devices

Martin Leucker

Institute for Software Engineering and Programming Languages
University of Lübeck
leucker@isp.uni-luebeck.de

Abstract. Medical devices, especially when operated in the surgery room, are safety critical systems, as the patient's life may depend on them. As such, there are high legal requirements to meet by the manufacturers of such devices. One of the typical requirements is that whenever medical devices are interconnected, the whole setup has to be approved by the corresponding legal body. For economical reasons, however, it is desirable to interconnect devices from different manufacturers in an individual fashion for each surgery room. Then however no integration test has been carried out a priori and thus the whole setup could not have been approved. In other words such economical demands impose challenges both on the technical as well as the legal situation. In this contribution, we report on these challenges as well as on first ideas to address them.

1 The Quest

The medical and health treatment of patients becomes more and more a highly technological procedure by sophisticated machines. Especially operations are carried out with the help of a collection of different, specialized devices. In Figure 1, a typical situation in a brain surgery is shown. The doctor is using an electronically enriched microscope to cut with an hf-intersector parts of some meningeom located in the patients brain. The position of the tumor was first identified using a device shown in the back of the picture. As expected, the patient is connected both to a ventilator as well as to an anesthetic machine during the whole operation. Most of the machines log their respective data for later use and are thus initialized with the main data of the patient.

All of these machines perform safety critical tasks and are as such safety critical systems. Each of them has to be approved by a legal body before it may be used in the operation room. The only exception is the application of individual machines approved by the doctor.[1]

To assure the safety of medical devices, they have to be developed following internationally accepted norms. For software of medical devices, for example, it has

[1] In this paper, we loosely follow the legal situation in Europe, which is similar, at least on the high-level discussion carried in this paper, to that in the US. Note that the aim of this paper is to give high-level account to the field but not a detailed and precise description of the legal situation.

Fig. 1. Snapshot from a brain surgery

to be developed using a predefined process model, for example the V-model or agile approaches. Moreover a dedicated risk analysis has to be carried out to identify the potential hazards the system under scrutiny may cause. Last but not least, dedicated verification of the system, typically carried out by intensive testing, has to be done and checked by the legal body. See [1], [2] and [3] for European directives that should be followed via following the corresponding national laws, e.g. the *Medizingproduktegesetz* in Germany, and, [4] for a more elaborate description of the legal situation in Europe when building medical devices.

Steadily rising health costs call for optimization also in the surgery room. One axis for optimization is the interconnection of medical devices. Then, patients data can be shared among the different machines. Moreover, for example, the hf-intersector may be controlled via the microscope allowing the doctor to operate two machines with a single interface. This would enable him or her to operate in a more convenient and thus also more efficient fashion.[2]

A technically similar but legally different setting is to interconnect devices to create systems with a new functionality. For example, using an electronically adjustable infusion pump together with suitable sensors might turn the whole system into an autonomous system acting as an anestletic machine. Technically, the underlying machines are also just connected. However, a *new* system is created, for which the risks it may cause in the applied situation are unclear a priori.

2 The Challenges

Whenever medical devices are connected to form a single device in the operation room, the whole setup has to be approved, typically by a legal body.[3] In

[2] In Germany, there is a huge effort to foster such interconnection of medical devices within the public funded project OR.NET. See `http://www.ornet.org` and [5] for further details.

[3] There is also the possibility of a self-approval by a doctor or a hospital. However, this is often only carried out in highly advanced clinics.

practice, this implies that usually machines are not interconnected at all as it is to cumbersome to approve a large number of different setups. The only exception is the possible interconnection of devices from a single manufacturer as the manufacturer may use this as an argument to promote its devices. Both for financial and technical reasons, it would however be desirable to interconnect devices from different manufacturers, allowing the hospital to select the devices with the best values for each individual task.

The main problems to overcome for reaching this goal, is risk management and the verification of the whole setup of devices.

3 The Approach—And Further Challenges

As each device is built and approved separately, a risk analysis has been carried out for each device. However, it seems not possible to assess the risk of a combined system plainly by looking a the risk of each individual system. In the situation studied in this paper, however, some modular risk analysis seems possible. Each device has its own, predefined functionality which may be enhanced by information/functionality provided by some other device. Assuming the safe operation of this other devices, it may be analyzed a priori, in which way the other machine influences the risks of the studied machine.

Nevertheless, a formal study on risks and when an how to reason about risk in a modular fashion is desirable, in general, and for the application in this setting of interconnected medical devices.

Testing of the whole setup of different devices has not been carried out by the manufacturers of the devices but each device has been tested individually. Looking at the complete setup, a *unit* test but no *integration* test has be performed. Thus, it would be desirable to perform an integration test dynamically at runtime.

To this end, we propose the following scheme:

- Each device comes with two operation modes. One is a *stand-alone mode* while the other mode is used when connected with other devices
- The connection of devices goes via precisely specified interfaces. Such a specification defines pre and post conditions for the values exchanged via the interface but also temporal requirements of the sequence of data transmitted via the interface. The latter implies, for example, that some communication protocol to be performed via the interface is precisely stated in the interface specification. See [6] for corresponding initial work in this direction.
- From the interface specification, monitoring devices may be synthesized using runtime verification techniques [7]. Such monitors may be used to check the data exchange of the connected devices.

A system of systems may then work as follows:

- When a device is not connected to any other device, it works in its stand-alone mode.

- When a device is connected to some other device, the compatibility of the interface specifications has to analyzed, as for example in [8]. If the two systems may not work together according to their specification, both work in the stand-alone mode. Otherwise, they work in the connected mode. Work on mediator synthesis may enhance the interconnection of devices [9].
- Whenever a system works in the connected mode, it uses the automatically synthesized monitors to check whether each other device follows its interface specification. Whenever a failure is detected, each system may fall back to its stand-alone mode.

It has to be investigated in which way the huge body of work on interface specifications, such as interface automata, as well as the findings in the field of runtime verification and runtime reflection [10] may be used with possibly adaption to realize the safe operation of connected devices in the surgery room.

References

1. Directive 93/42/EEC: Council Directive 93/42/EEC of 14 June 1993 concerning medical devices, OJ L 169 of 12 July 1993
2. Directive 98/79/EC: Directive 98/79/EC of the European Parliament and of the Council of 27 October 1998 on in vitro diagnostic medical devices, OJ L 331 of 7 December 1998
3. Directive 90/385/EEC: Council Directive 90/385/EEC of 20 June 1990 on the approximation of the laws of the Member States relating to active implantable medical devices, OJ No L 189 of 20 July 1990
4. Johner, C., Hölzer-Klüpfel, M., Wittorf, S.: Basiswissen Medizinische Software: Aus- und Weiterbildung zum Certified Professional for Medical Software. Dpunkt (2011)
5. Kühn, F., Leucker, M.: Or.net: Safe interconnection of medical devices - (position paper). In: Gibbons, J., MacCaull, W. (eds.) FHIES 2013. LNCS, vol. 8315, pp. 188–198. Springer, Heidelberg (2014)
6. de Alfaro, L., Henzinger, T.A.: Interface automata. In: ESEC/SIGSOFT FSE, pp. 109–120. ACM (2001)
7. Leucker, M., Schallhart, C.: A brief account of runtime verification. Journal of Logic and Algebraic Programming 78(5), 293–303 (2009)
8. Chakrabarti, A., de Alfaro, L., Henzinger, T.A., Jurdziński, M., Mang, F.Y.C.: Interface compatibility checking for software modules. In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, pp. 428–441. Springer, Heidelberg (2002)
9. Bennaceur, A., Chilton, C., Isberner, M., Jonsson, B.: Automated mediator synthesis: Combining behavioural and ontological reasoning. In: Hierons, R.M., Merayo, M.G., Bravetti, M. (eds.) SEFM 2013. LNCS, vol. 8137, pp. 274–288. Springer, Heidelberg (2013)
10. Leucker, M.: Checking and enforcing safety: Runtime verification and runtime reflection. ERCIM News 2008(75) (2008)

Temporal Logic Based Monitoring of Assisted Ventilation in Intensive Care Patients

Sara Bufo[1], Ezio Bartocci[2], Guido Sanguinetti[3,4], Massimo Borelli[1], Umberto Lucangelo[5], and Luca Bortolussi[1,6,7]

[1] Department of Mathematics and Geosciences, University of Trieste, Italy
[2] Faculty of Informatics, Vienna University of Technology, Austria
[3] School of Informatics, University of Edinburgh, UK
[4] SynthSys, Centre for Synthetic and Systems Biology, University of Edinburgh, UK
[5] Department of Medicine, University of Trieste, Italy
[6] Computer Science Department, Saarland University, Saarbrücken, Germany
[7] CNR/ISTI, Pisa, Italy

Abstract. We introduce a novel approach to automatically detect ineffective breathing efforts in patients in intensive care subject to assisted ventilation. The method is based on synthesising from data temporal logic formulae which are able to discriminate between normal and ineffective breaths. The learning procedure consists in first constructing statistical models of normal and abnormal breath signals, and then in looking for an optimally discriminating formula. The space of formula structures, and the space of parameters of each formula, are searched with an evolutionary algorithm and with a Bayesian optimisation scheme, respectively. We present here our preliminary results and we discuss our future research directions.

1 Introduction

Temporal logic (TL) has proved to be a powerful and natural framework to describe complex temporal properties of systems. In fact, temporal logic formulae describe temporal patterns between events in a form which is close to our way of thinking, and as such they are intelligible and suitable to represent behavioral specifications. The availability of efficient verification and monitoring algorithms, that can check if a property is satisfied by a model or an observed run of a system, has further fostered this logical approach as a tool for design.

Monitoring, in particular, is applied mainly to engineered systems, as TL can naturally be used to encode the desired behavioural specifications the system should satisfy, which are provided by the designer [17]. The availability of software and hardware for real time verification, however, makes this approach very attractive also in medicine, for instance to monitor the ECG signal or the flow/pressure curves of assisted ventilation of a patient in intensive care.

The main obstacle in this respect is that the behavioural specification we should observe are unknown: how can we describe by a TL formula the emergence of a dangerous clinical condition? The experience of physicians can help us identify situations in which the observed signals are prodrome to the insurgence of clinical complications, but a precise characterisation of these conditions in TL is by no means easy to obtain. Such a description would enable practitioners to use available monitoring tools, constructing devices that can support physicians in critical care choices.

The alternative to the unfeasible manual derivation of such specifications is to learn TL formulae from observed data, in the form of (manually) annotated signals. For instance, we can have as input from physicians a set of flow/pressure curves in which abnormal respiratory acts have been identified. Learning a TL specification of such abnormalities essentially means to construct a TL classifier of signals, that can separate normal breaths from critical ones. An appealing aspect of classifying by TL formulae would be the ease of interpretation of results, and the possibility of obtaining actionable physiological insights from the classifier. While statistical classifiers such as support vector machines often achieve impressive accuracy, this comes at the cost of developing opaque non-linear maps which offer little in the way of physiological insight.

Learning TL specifications from data is a problem that has recently received a certain attention in the literature [2,12,19,40,41,24], and which will be discussed in the related work section (Section 5). A frequently encountered problem with these approaches is the very large amount of data needed to learn inductively properties which are robust to noise in the observations [2,24]. The approach we consider here, which has been introduced in [4,5], tries to recast the learning problem within a solid statistical framework. Our strategy, instead, is to first infer a generative statistical model of the observed data, and then learn temporal specifications that have a high probability of being true in the so obtained model. This naturally keeps the effects of noise under control in a systematic way, but also solves the data shortage problem, as we can generate as much synthetic data as needed. In this work we consider a variant of this learning problem in which we aim at distinguishing two sets of signals, the good and the bad ones. This is obtained by constructing a statistical model for each class of signals, and then assigning to each TL formula a score which is high when the formula is true with high probability in a model and false with high probability in the other one.

From a medical perspective, we have started applying this framework to the identification of respiratory problems in patients in intensive care, which are breathing under assisted ventilation. In particular, our goal is to classify single respiratory acts into normal and abnormal. In principle, we want to look for different types of abnormality, although at this stage we focussed on ineffective triggering efforts, i.e. on the asynchrony between the flow of the ventilator and the attempt of the patient to start a new breath. Although a single occurrence of such event per se is not dangerous, and as such is largely ignored in practice, a long sequence of them can lead to severe clinical complications. This, and the fact that most ventilators in the market are not equipped with monitoring routines, motivates the investigation of this problem. More details, also from a biological and clinical perspective, will be given in Section 2.

In Section 3, instead, we discuss the basic steps of our methodological approach, namely the construction of statistical generative models of the signals we consider, which here take the form of a Stochastic Hybrid System (Section 3.1), the TL we use, which is the time-bounded fragment of Metric Temporal Logic (Section 3.2), the procedure to learn the structure of TL classifiers (based on an Evolutionary Algorithm, Section 3.4), and the method to learn the best formula parameters, based on Bayesian optimisation (Section 3.5). Some results are presented in Section 4, while conclusions will be drawn in Section 6.

2 Assisted Ventilation and Patient Ventilator Asynchronies

Pulmonary ventilation is the process of air flowing into and out of the lungs and occurs because the pressure of the atmosphere and of the gases inside the lungs differ.

During inspiration, the diaphragm and the external intercostal muscles contract, leading to an increase in volume of the thoracic cavity. As a result, the pressure within the lungs decreases and falls below atmospheric pressure and air flows into the lungs.
On the contrary, in the expiratory phase the relaxation of the diaphragm decreases the thoracic volume and the sign of the pressure gradient changes (becomes positive), causing the direction of flow to be reversed. As air moves when breathing is accomplished, oxygen gas and carbon dioxide are exchanged.

In patients suffering from acute respiratory failure, such gas exchange is inadequate and normal pulmonary ventilation is augmented or replaced by a mechanical ventilator. Mechanical ventilators are machines that generate a controlled flow of gas and constantly measure the airway pressure (Paw), the quantity of air that enters the lungs per unit time (flow Q) and how much air enters and leaves the lungs (volume V). Despite the possibility of continuously monitoring such ventilatory parameters (Paw, Q and V), one of the major clinical concerns in mechanical ventilation is represented by asynchronies, a generic term describing a wide class of 'poor interactions' between the mechanical ventilator and human breathing. Asynchronies affect more than one third of mechanically ventilated patients [35,37,13] and, despite the debated question of cause-effect relation to poor outcome [10], they generate stress and discomfort for the patient, providing uncontrolled delivery of large volumes or high pressures to the patient respiratory system. Asynchronies potentially contribute to ventilator induced lung injury [39,36,30,38,23]. Asynchronies can appear during all the phases of the respiration: the triggering phase; the pressure-delivery phase; the cycling-off phase [22]. During the initial triggering phase, triggering delay, ineffective inspiratory effort and auto-triggering may occur. During the pressure-delivery phase the ineffective triggering is the major concern, but also inadequate or excessive ventilator assist is a problem, as well as the lack for an optimal setting of pressure rise time. During the cycling-off phase the premature opening (inadequate assist and double triggering) and the late opening (triggering delay and ineffective effort) of the expiratory valves are the major concerns [36,23,22]. Asynchronies can also interact appearing into one breathing act [27]. Only sophisticated ventilators are currently equipped with supplementary devices (e.g. neurally adjusted ventilation, [32]) that reveal and quantify [31] the presence of such phenomena. A human intervention is therefore often required to analyse and interpret data. For this reason, simple algorithms based on standard waveforms of pressure, flow and volume able to detect anomalies will be useful tools to automatise the diagnostic process [22]. Currently, various algorithm have been presented. While in [15] the ineffective inspiration triggering efforts have been addressed by a FORTRAN procedure evaluating phase portrait flow loops, other authors [13,28,29,8] directly investigate on numerical or analytical aspects of flow Q waveform.

In this context, two problems call for consideration, i.e. the classification of single breathing acts and the recognition of sequences of breaths exhibiting a pattern leading to severe respiratory failure.

Learning logical formulae discriminating between different conditions is a possible line of research in both cases and could be easily put into practice implementing monitoring algorithms in cheap hardware such as FPGA-based devices.

In this paper the focus is set on the first problem. In particular, we are interested in learning temporal logic properties that characterise single breathing acts. The methodology that we illustrate is then applied to a specific case, i.e. the recognition of ineffective inspiratory efforts considering flow data. An ineffective inspiratory effort (IE) is a condition that arises when a patient receiving mechanical ventilation tries to inspirate when the pressure gradient is positive and the drop in pressure related to the activation of the inspiratory muscles is unable to change the sign of the gradient, causing inspiration and triggering of a new ventilation cycle not to occur. A single breath may be affected by one or more IE and the presence of each IE may be revealed by the presence of a hump in the flow curve, see Figure 1.

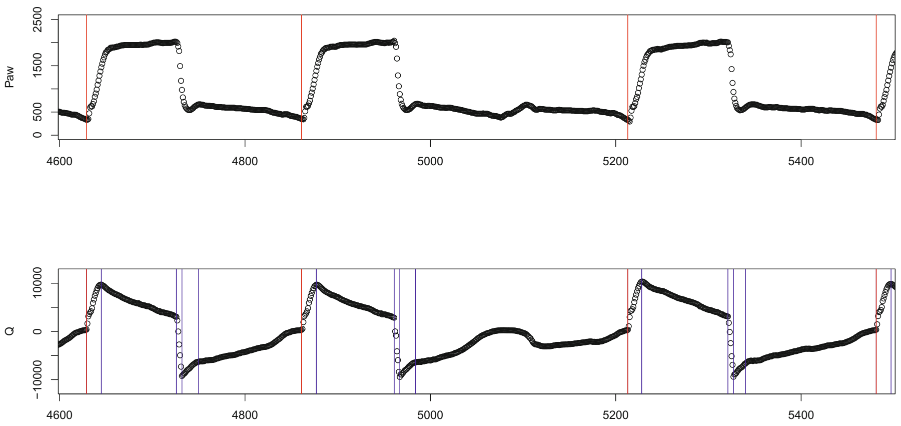


Fig. 1. *Paw* and *Q* tracings of two standard breaths and a breathing act with an IE divided into single respiratory acts (red lines). The different phases used to build the stochastic models of flow curves (Section 3.1) are also highlighted (blue lines).

3 Methodology

The general problem of learning temporal properties of a system $\mathcal{A}$ can be rephrased and recast within specific contexts, in accordance with the available data and the final objective. We assume that the system is observable and system observations are available and conceive properties as logical statements. Within this framework, we consider a discriminative variation of the learning problem, i.e. a second system $\mathcal{B}$ is introduced and properties that best discriminate between $\mathcal{A}$ and $\mathcal{B}$ (i.e. logical formulae that are satisfied by $\mathcal{A}$ and not by $\mathcal{B}$) are searched for. Different approaches to this problem are possible. At a high level, our methodology starts by devising a data-driven statistical abstraction of each system. In this way, systems are represented by generative models which can be simulated *ad libitum* (preventing the occurrence of data shortage problems) and properties describe the trajectories sampled from the models. The second step is the property synthesis phase, where learning of formulae is performed. In more detail, a score function $R(\varphi)$, depending on the formula φ, based on the simulation of both models and representative of the discriminating power of each formula is introduced and optimised. Even though other choices are possible, we have decided to consider structure and parameter formulae components separately and tackle these suboptimisation problems using a local search algorithm and a Bayesian optimisation approach, respectively.

In the following sections, the methodology introduced above will be applied to learn properties of flow curves of MV breathing acts with an ineffective effort (IE, system $\mathcal{A}$). In order to capture properties that are related to the IE only, standard breath flow curves are considered as system $\mathcal{B}$. The statistical models used to represent $\mathcal{A}$ and $\mathcal{B}$ are Stochastic Hybrid Systems (Section 3.1) and the logic chosen to specify properties is $\mathrm{MITL}_{[a,b]}$ (Section 3.2). The score function $R(\varphi)$ is based on the log odds ratio (Section 3.3) and is optimised considering structure and parameter formulae components separately. Structural learning is accomplished with an Evolutionary Algorithm (Section 3.4) and formula parameters are refined resorting to a Bayesian optimisation routine (Section 3.5).

3.1 Statistical Modelling of Ventilation Signals

The models chosen to represent flow curves are Stochastic Hybrid Systems [11,16] devised from clinical data of a patient assisted through mechanical ventilation. The training data used to build our models were organised as discrete time series and sampled flow values of 46 breathing acts with an IE in the expiration phase ($\mathcal{A}$ training data) and 251 standard breaths ($\mathcal{B}$ training data), from a single patient. We will now briefly illustrate how the model of system $\mathcal{B}$ was built. Then, we will explain how we derived the model of system $\mathcal{A}$ from this. Looking at the flow curve structure, we can see that each breath can be naturally divided into five parts or phases (see Figures 1 and 2). Within each phase, representing the discrete skeleton of the hybrid model, we described the evolution by continuous components representing the flow value and the duration of the phase. Time is kept discrete, to mimic the sampling frequency of real data, equal to 100Hz. We supposed that the length of each phase was normally distributed, and devised mean and variance parameters of each discrete sub-model from training data. A new duration (truncated to the closer time step) is sampled every time the system changes phase, hence this operation can be formally modelled as part of the reset function attached to each discrete transition. The flow component was instead treated as a discrete dynamical model: the flow value at a time instant t is a function of the flow value at the time instant $t-1$. Visual inspection of normal patient traces (shown in e.g. Figure 2) suggested that, within each breathing phase, a linear first-order autoregressive model may be appropriate. The resulting model of system $\mathcal{B}$ is therefore

$$\begin{cases} flow(t+1) = f_k(flow(t)) + \varepsilon_k \\ f_k(flow(t)) = a_k \cdot flow(t) + b_k \\ \varepsilon_k \sim \mathcal{N}(0, \alpha_k^2) \\ length_k \sim \mathcal{N}(\mu_k, \sigma_k^2) \end{cases}$$

Parameters a_k, b_k and α_k are calculated using linear regression from $\mathcal{B}$ training data. Whereas for $k = 1, \dots, 4$ the slope a_k and the intercept b_k are constant, for $k = 5$ these coefficients depend on the length (i.e. on the realization of $length_5$). This choice is based on the observation that length, intercept and slope of final parts are highly correlated.

The model of system $\mathcal{A}$ was built in a similar way from the correspondent training data. Inspection of sample IE trajectories revealed a conspicuous anomaly in phase 5 of the breathing act; we therefore introducted a novelty factor in the phase 5 submodel to capture the presence of the IE. We decided to tackle this part introducing a hierarchical model, i.e. to describe an IE signal as a normal signal plus a perturbation, which for IE in the expiration phase is a sinusoidal-like hump, see Figure 3.1. We constructed a statistical model of such a hump by fitting a polynomial curve (whose degree, equal to seven, was selected by optimising the Aikake information content [7]).

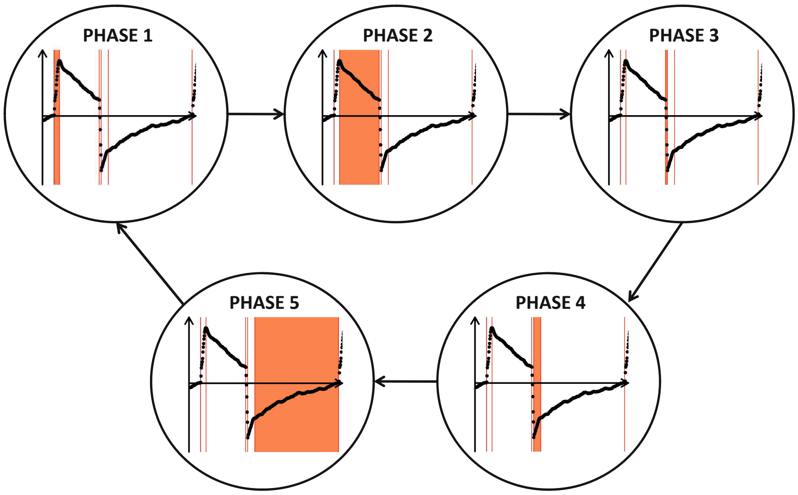


Fig. 2. Scheme of the stochastic hybrid model of standard breath flow curves. Each phase corresponds to the highlighted segment of the flow curve.

3.2 Metric Interval Temporal Logic

We consider here MITL$_{[a,b]}$ [1,26] a fragment of the Metric Temporal Logic [25] with linear time-bounded temporal operators which has proven to be an efficient formalism to characterise properties of real-valued signals evolving in continuous time. For this reason, we have decided to adopt this logic to specify properties of the trajectories sampled from our models.

The syntax of MITL$_{[a,b]}$ is given by the following grammar:

$$\varphi ::= \top \mid \mu \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \mathrm{U}_{[a,b]} \varphi_2$$

where $\top$ is the true formula and temporal modalities are restricted to intervals of the form $[a,b]$ with $0 \leq a < b$ and $a, b \in \mathbb{Q}_{\geq 0}$. Formulae are built from atomic propositions μ using boolean operators $\neg$, $\wedge$ and time-constrained versions of the until operator

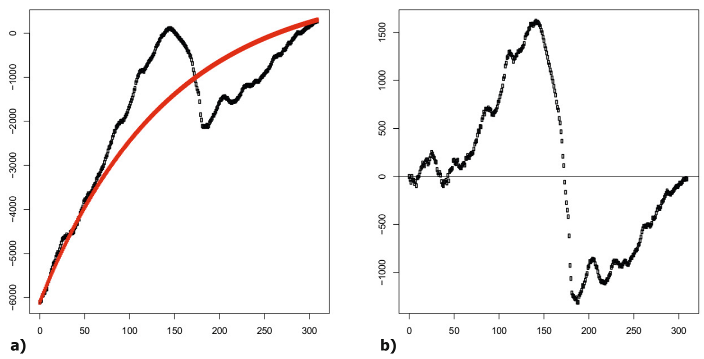


Fig. 3. (a) Flow signal f (part 5, black) of a breathing act with an IE with overlapped the normal signal n (red). (b) Perturbation of the signal , computed as $f - n$.

U. Atomic propositions are boolean predicate transformers, i.e. operators transforming real-valued functions into boolean signals, which provide a true ($\top$) or false ($\perp = \neg\top$) value to the formula at each time instant.

Further temporal modalities are derived from the MITL$_{[a,b]}$ syntax and commonly used. As an example, time-bounded eventually $\Diamond_{[a,b]}\varphi \equiv \top\ \mathrm{U}_{[a,b]}\ \varphi$ and time-bounded globally $\Box_{[a,b]}\varphi \equiv \neg\Diamond_{[a,b]}\ \neg\varphi$ can be defined as usual from the until operator. MITL$_{[a,b]}$ formulae are interpreted over a time instant t and a real-valued function $\mathbf{x}$, and the satisfaction relation is given in a standard way, see e.g. [26]. We recall that a stochastic model induces a distribution on the space of trajectories, hence we can compute the probability of the set of trajectories that satisfies a given MITL$_{[a,b]}$ formula φ. We will refer to such probability $p(\varphi)$ as the satisfaction probability of φ, see e.g. [3] for further details. In the context of this work, we estimated such a probability by statistical means, resorting to Statistical Model Checking [14,42].

3.3 Discrimination Function

The problem of finding formulae that are likely to be satisfied by trajectories sampled from the model of system $\mathcal{A}$ but not by trajectories sampled from the model of system $\mathcal{B}$ is translated into an *optimisation problem* of the *discrimination function* $R(\varphi)$ associated with a MITL$_{[a,b]}$ formula φ. A possible choice for such function is $R(\varphi) = L(\varphi)$, the log odds ratio between the satisfaction probabilities

$$L(\varphi) = \log \frac{p(\varphi \mid \mathcal{A}\ \text{model})}{p(\varphi \mid \mathcal{B}\ \text{model})} \tag{3.1}$$

In this case, penalty terms could be introduced to favour formulae which satisfy certain properties (e.g. thus penalising complex formulae over simple ones).

3.4 Structural Learning

We will now present how the structure of the discriminating MITL$_{[a,b]}$ formulae (i.e. the formulae which optimise φ) was found. As previously mentioned, we decided to tackle this optimisation problem using an Evolutionary Algorithm (EA) [20]. EAs are a class of search and optimisation algorithms inspired by models of the natural selection of species. The main idea of an EA is to consider a starting (usually randomly chosen) population of candidate solutions (the starting generation) and iteratively evolve it towards better solution sets. Each iterative step produces a new generation by manipulating the previous one using stochastic operators (the genetic operators) and the procedure ends when a fixed number of generations has elapsed or some form of convergence criterion has been met. The most simple EAs are based on the use of three genetic operators which resemble the biological principles of survival of the fittest (selection operator), reproduction (recombination operator) and gene mutation (mutation operator). In the framework of EAs, selection is used to choose the individuals (parents) that will pass the information they contain to the next generation, recombination to generate new (and possibly better) individuals by combining parental individuals information and mutation to introduce innovation in the population. One of the main attraction of EAs is that operators are practically implemented by simple algorithms and usually finds very quickly good solutions.

When learning discriminating formulae in our case study, we considered populations of MITL$_{[a,b]}$ formulas, represented by their parsing trees. Within this framework, recombination and mutation are simply implemented by performing with a certain probability an exchange of parental subformulas (recombination) and a modification of a node (mutation, e.g. of a boolean or temporal operator).

3.5 Parameter Learning

We now turn to the issue of tuning the parameters of formulae to maximise their satisfaction probability. More specifically, suppose to have a MITL$_{[a,b]}$ formula φ_θ which depends on some continuous parameters θ. We aim to maximise its discriminative power $R(\varphi_\theta)$ defined in equation (3.1). Naturally, this quantity is an intractable function of the formula parameters; yet its value at a finite set of parameters can be noisily estimated using a stochastic model checking procedure, i.e. by simulating the model for a certain number n of times, checking the formula in each run, and then estimating $R(\varphi_\theta)$ from the so generated data. The problem is therefore to identify the maximum of an intractable function with as few (approximate) function evaluations as possible. This problem is closely related to the central problem of reinforcement learning of determining the optimal policy of an agent with as little exploration of the space of actions as possible. We therefore adopt a provably convergent stochastic optimisation algorithm, the GP-UCB algorithm [33], to solve the problem of continuous optimisation of formula parameters. Intuitively, the algorithm interpolates the noisy observations using a stochastic process (a procedure called emulation in statistics) and uses the uncertainty in this fit to determine regions where the true maximum can lie. This algorithm has already been used in a formal modelling scenario in [9].

4 Results: Monitoring Ineffective Respiratory Acts

We present here the results obtained by applying our learning procedure on discrimination of IE occurring during expiration. Taking into account only the expiratory phase, only the last part of the trajectories sampled from the statistical models (i.e. phase 5) is considered. Accordingly, the time instant 0 of the $\text{MITL}_{[a,b]}$ formulae refers to the time instant when phase 5 is entered. The set of $\text{MITL}_{[a,b]}$ formulae examined is built over the set of atomic propositions

$$\mathcal{P} = \{flow \leq \lambda\} \cup \{flow \geq \lambda\} \cup \{flow' \leq \mu\} \cup \{flow' \geq \mu\}$$

where $flow'(t) = flow(t+1) - flow(t)$. We search for short formulae maximising the discrimination function $R(\varphi)$ associated with a $\text{MITL}_{[a,b]}$ formula φ, described in 3.3. Since a trajectory sampled from a statistical model does not have a fixed duration (it is thus not always possible to know *a priori* if its truth value over a $\text{MITL}_{[a,b]}$ formula φ is definable), a penalty term $U(\varphi)$ is introduced to keep track of the number of non-sufficiently long trajectories generated during the calculation of the value of φ over φ. As a result, $R(\varphi) = L(\varphi) - S(\varphi) - U(\varphi)$, where $L(\varphi)$ is the log odds ratio between the satisfaction probabilities and $S(\varphi)$ is a size penalty. We experimented our learning algorithm by testing different parameters and settings, such as different variants of the Evolutionary Algorithm operators, the frequency of utilisation of GP-UCB within the evolutionary algorithm (i.e., we optimised all elements of a population, only best candidate solutions, only best solutions at the end of the algorithm), and the values of the penalty terms. The best formulae obtained are

$$\varphi_1 = \Box_{[0.4518,0.8609]}(\Diamond_{[0.7853,0.9394]}(\Box_{[0.6370,0.8222]}(\Diamond_{[0.7923,0.8070]}(flow \geq -4554.0))))$$
$$\varphi_2 \equiv \Diamond_{[0.3966,1.6705]}(flow' \leq -144.2708)$$

Their satisfaction probabilities $p_{\mathcal{A}}(\varphi) = p(\varphi \mid \mathcal{A} \text{ model})$ and $p_{\mathcal{B}}(\varphi) = p(\varphi \mid \mathcal{B} \text{ model})$, summarised in the table below, were estimated by statistical model checking [42,14].

	φ_1	φ_2
$p_{\mathcal{A}}$	0.5040	0.88523
$p_{\mathcal{B}}$	$< 10^{-3}$	$< 10^{-3}$

If we inspect these two formulae, we can easily understand their meaning. Formula φ_1 roughly forces the signal to be longer than 3 seconds (forcing the flow to be defined at that time), and captures the fact that IE respiratory acts tends to last longer than normal ones. Formula φ_2, instead, detects a quick drop in the flow, corresponding the decreasing part of the hump, which is generally not present in a normal breath.

Formulae were then validated on real data from the same patient considered in the training phase, specifically on a test set of 345 standard breaths and 77 breathing acts with an IE. In this phase, φ_1 was able to recognise 33 ineffective efforts, whereas φ_2 26. False positives (i.e. normal breaths satisfying formulae) were detected during validation of φ_1 only. We decided to merge these two formulae using logical disjunction and validate the obtained formula $\varphi_1 \vee \varphi_2$. As a result, 58 ineffective efforts (75.3%) and 336 standard breaths (97.4%) were correctly classified.

5 Related Work

Mining temporal logic specifications from data is an emerging field of computer aided verification [2,5,12,19,40,41]. This task usually depends on the availability of a fully specified model, enabling a quantitative evaluation of the probability that a certain formula will hold. Machine learning techniques, such as decision trees [19] or stochastic optimisation methods [41,40] can be then employed to improve the confidence with which the formula will be satisfied.

Learning temporal logic specifications directly from observed traces of the system remains a more challenging problem. In general, solving the full structure and parameter learning problem is infeasible, due to the intractability resulting from a hybrid combinatorial/continuous optimisation problem. Heuristic search approaches have been proposed in [12]; while these may prove effective in specific modelling problems, they generally do not offer theoretical guarantees, and can be prone to over-fitting/vulnerable to noise. Geometric approaches such as the one proposed in [2] rest on solid mathematical foundations but can also be vulnerable to noise, and require potentially very large amounts of data to permit identification. The work of [24], instead, employs a notion of robustness of satisfiability of a formula to guide an optimisation based mining procedure. While this approach can be applied also in a model-free scenario, empirical estimation of the robustness of a formula may require the observation of a large number of traces of the system. Furthermore, the approach is based on some monotony properties of a subset of formulae which does not hold for the log-odd ratio score.

Our approach instead combines statistical modelling ideas from machine learning with formal verification methods. In this respect, our work is related to a number of other recent attempts to deploy machine learning tools within a verification context [5,6,34,21]. Similar ideas to the ones used in this paper have been deployed on the parameter synthesis problem in [5,9,3], where the GP-UCB algorithm was used to identify the parameters of a model which maximised the satisfaction/robustness of a formula. Statistical abstractions draw their roots in the *emulation* field in statistics: within the context of dynamical systems, emulation has been recently used in [18] to model compactly the interface between subsystems of complex gene regulatory networks.

6 Conclusions

We presented a method to learn temporal properties discriminating two classes of temporal signals. First, we derive the generative statistical models of the two sets. Then, we explore the formula space searching for good discriminating formulae with a combination of evolutionary algorithms and bayesian optimisation strategies. This method has been applied to detect ventilator asynchronies in patients in intensive case, and exemplified on the detection of ineffective respiratory efforts during expiration.

The method we presented is still in a preliminary development stage, and has some limitations. First of all, the trained formulae consider only the flow; keeping track of pressure should increase its performance. Secondly, the parameters of the formulae depend on properties of input signals like the range of the flow and the average phase duration, so that they tend to be patient specific. One way to attack this problem would be to optimise again the (key) parameters while starting monitoring a new patient. A more interesting alternative can be to normalise flow and pressure signals so that their duration and range becomes the same for any patient. We are currently investigating the benefits and limits of this idea. More generally, a difficulty we found is that the hard time bounds of formulae conflict with the different durations of breaths even for a single patient. Possible solutions we are investigating include adding more discrete phases to the generative models or checking properties of signals in the flow/pressure phase space, rather than of the time-flow/pressure representation.

Another issue with the current approach is the score function. The log odd ratio, in fact, tends to privilege the decrease of the satisfaction probability of the formula in the second model rather than its increase in the first one, i.e. to decrease the false positive rate rather then the false negative one. The reason for this is readily explained: if the probability in the second model passes from 10^{-3} to 10^{-2} then the log odd ratio decreases by an additive term of $-\log 10$, while if the satisfaction probability of the first model passes from 0.5 to 1, the log odd ratio in increased only by $\log 2$. Hence, better scoring function are needed. Indeed, this is confirmed by the following experiment with the formula φ_2 of Section 4: we run the GP-UCB algorithm optimising only its satisfaction probability in the first model, varying the threshold $\theta_0 \approx -144$ in the range $[-300, -30]$. In this case, the problem resulted monotonic and the optimum is obtained for $\theta^* = -30$. With this new parameter, the discriminative power of the formula φ_2 alone on the validation set passed from a false negative (false positive) rate of 60% (of 0%) to a rate of 8% (of 3.3%).

The presented method can be further extended in trying to detect other kinds of asynchronies and surely requires extensive testing before reaching one of our final goals, i.e. its implementation in a dedicated hardware.

Acknowledgements. L.B. acknowledges partial support from the EU-FET project QUANTICOL (nr. 600708) and by FRA-UniTS. G.S. acknowledges support from the ERC under grant MLCS306999. E.B. acknowledges the support of the IKT der Zukunft of Austrian FFG project HARMONIA (nr. 845631).

References

1. Alur, R., Feder, T., Henzinger, T.A.: The benefits of relaxing punctuality. J. ACM 43(1), 116–146 (1996)
2. Asarin, E., Donzé, A., Maler, O., Nickovic, D.: Parametric Identification of Temporal Properties. In: Khurshid, S., Sen, K. (eds.) RV 2011. LNCS, vol. 7186, pp. 147–160. Springer, Heidelberg (2012)
3. Bartocci, E., Bortolussi, L., Nenzi, L., Sanguinetti, G.: On the robustness of temporal properties for stochastic models. In: Proc. of HSB 2013, pp. 3–19 (2013)
4. Bartocci, E., Bortolussi, L., Sanguinetti, G.: Learning temporal logical properties discriminating ECG models of cardiac arrhytmias. CoRR abs/1312.7523 (2013)
5. Bartocci, E., Bortolussi, L., Sanguinetti, G.: Data-driven statistical learning of temporal logic properties. In: Legay, A., Bozga, M. (eds.) FORMATS 2014. LNCS, vol. 8711, pp. 23–37. Springer, Heidelberg (2014)
6. Bartocci, E., Grosu, R., Karmarkar, A., Smolka, S.A., Stoller, S.D., Zadok, E., Seyster, J.: Adaptive runtime verification. In: Qadeer, S., Tasiran, S. (eds.) RV 2012. LNCS, vol. 7687, pp. 168–182. Springer, Heidelberg (2013)

7. Bishop, C.M.: Pattern Recognition and Machine Learning. Springer (2006)
8. Blanch, L., Sales, B., Montanya, J., Lucangelo, U., Garcia-Esquirol, O., Villagra, A., Chacon, E., Estruga, A., Borelli, M., Burgueño, M., Oliva, J., Fernandez, R., Villar, J., Kacmarek, R., Murias, G.: Validation of the better care system to detect ineffective efforts during expiration in mechanically ventilated patients: A pilot study. Intensive Care Med. (in press)
9. Bortolussi, L., Sanguinetti, G.: Learning and Designing Stochastic Processes from Logical Constraints. In: Joshi, K., Siegle, M., Stoelinga, M., D'Argenio, P.R. (eds.) QEST 2013. LNCS, vol. 8054, pp. 89–105. Springer, Heidelberg (2013)
10. Branson, R.: Patient-ventilator interaction: The last 40 years. Respir. Care 56(1), 15–24 (2011)
11. Bujorianu, M.L., Lygeros, J.: General stochastic hybrid systems. In: IEEE Mediterranean Conference on Control and Automation MED, vol. 4, pp. 1872–1877 (2004)
12. Calzone, L., Chabrier-Rivier, N., Fages, F., Soliman, S.: Machine learning biochemical networks from temporal logic properties. In: Priami, C., Plotkin, G. (eds.) Trans. on Comput. Syst. Biol. VI. LNCS (LNBI), vol. 4220, pp. 68–94. Springer, Heidelberg (2006)
13. Chen, C., Lin, W., Hsu, C., Cheng, K., Lo, C.: Detecting ineffective triggering in the expiratory phase in mechanically ventilated patients based on airway flow and pressure deflection: Feasibility of using a computer algorithm. Crit. Care Med. 36(2), 455–461 (2008)
14. Clarke, E., Donzé, A., Legay, A.: On simulation-based probabilistic model checking of mixed-analog circuits. Formal Methods in System Design 36(2), 97–113 (2010)
15. Cuvelier, A., Achour, L., Rabarimanantsoa, H., Letellier, C., Muir, J., Fauroux, B.: A noninvasive method to identify ineffective triggering in patients with noninvasive pressure support ventilation. Respiration 80(3), 198–206 (2010)
16. Davis, M.: Markov Models and Optimization. Chapman & Hall (1993)
17. Donzé, A., Maler, O., Bartocci, E., Nickovic, D., Grosu, R., Smolka, S.: On temporal logic and signal processing. In: Chakraborty, S., Mukund, M. (eds.) ATVA 2012. LNCS, vol. 7561, pp. 92–106. Springer, Heidelberg (2012)
18. Georgoulas, A., Clark, A., Ocone, A., Gilmore, S., Sanguinetti, G.: A subsystems approach for parameter estimation of ode models of hybrid systems. In: Proc. of HSB 2012. EPTCS, vol. 92 (2012)
19. Grosu, R., Smolka, S.A., Corradini, F., Wasilewska, A., Entcheva, E., Bartocci, E.: Learning and detecting emergent behavior in networks of cardiac myocytes. Commun. ACM 52(3), 97–105 (2009)
20. Hoos, H.H., Stützle, T.: Stochastic local search: Foundations & applications. Elsevier (2004)
21. Kalajdzic, K., Bartocci, E., Smolka, S.A., Stoller, S.D., Grosu, R.: Runtime Verification with Particle Filtering. In: Legay, A., Bensalem, S. (eds.) RV 2013. LNCS, vol. 8174, pp. 149–166. Springer, Heidelberg (2013)
22. Kondili, E., Akoumianaki, E., Alexopoulou, C., Georgopoulos, D.: Identifying and relieving asynchrony during mechanical ventilation. Expert Rev. Respir. Med. 3(3), 231–243 (2009)
23. Kondili, E., Prinianakis, G., Georgopoulos, D.: Patient-ventilator interaction. Br. J. Anaesth. 91(1), 106–119 (2003)
24. Kong, Z., Jones, A., Ayala, A.M., Gol, E.A., Belta, C.: Temporal Logic Inference for Classification and Prediction from Data. In: Proc. of HSCC 2014 (2014)
25. Koymans, R.: Specifying real-time properties with metric temporal logic. Real-Time Syst. 2, 255–299 (1990)
26. Maler, O., Nickovic, D.: Monitoring temporal properties of continuous signals. In: Lakhnech, Y., Yovine, S. (eds.) FORMATS/FTRTFT 2004. LNCS, vol. 3253, pp. 152–166. Springer, Heidelberg (2004)
27. Mellott, K., Grap, M., Munro, C., Sessler, C., Wetzel, P., Nilsestuen, J., Ketchum, J.: Patient ventilator asynchrony in critically ill adults: Frequency and types. Heart Lung 43(3), 231–243 (2014)

28. Mulqueeny, Q., Ceriana, P., Carlucci, A., Fanfulla, F., Delmastro, M., Nava, S.: Automatic detection of ineffective triggering and double triggering during mechanical ventilation. Intensive Care Med. 33(11), 2014–2018 (2007)
29. Mulqueeny, Q., Redmond, S., Tassaux, D., Vignaux, L., Jolliet, P., Ceriana, P., Nava, S., Schindhelm, K., Lovell, N.: Automated detection of asynchrony in patient-ventilator interaction. In: Conf. Proc. IEEE Eng. Med. Biol. Soc., pp. 5324–5327 (2009)
30. Sassoon, C., Foster, G.: Patient-ventilator asynchrony. Curr. Opin. Crit. Care 7(1), 28–33 (2001)
31. Sinderby, C., Liu, S., Colombo, D., Camarotta, G., Slutsky, A., Navalesi, P., Beck, J.: An automated and standardized neural index to quantify patient-ventilator interaction. Critical Care 17, 239 (2013)
32. Sinderby, C., Navalesi, P., Beck, J., Skrobik, Y., Comtois, N., Friberg, S., Gottfried, S.B., Lindström, L.: Neural control of mechanical ventilation in respiratory failure. Nat. Med. 5(12), 1433–1436 (1999)
33. Srinivas, N., Krause, A., Kakade, S.M., Seeger, M.W.: Information-theoretic regret bounds for gaussian process optimization in the bandit setting. IEEE Transactions on Information Theory 58(5), 3250–3265 (2012)
34. Stoller, S.D., Bartocci, E., Seyster, J., Grosu, R., Havelund, K., Smolka, S.A., Zadok, E.: Runtime Verification with State Estimation. In: Khurshid, S., Sen, K. (eds.) RV 2011. LNCS, vol. 7186, pp. 193–207. Springer, Heidelberg (2012)
35. Thille, A., Rodriguez, P., Cabello, B., Lellouche, F., Brochard, L.: Patient-ventilator asynchrony during assisted mechanical ventilation. Intensive Care Med. 32(10), 1515–1522 (2006)
36. Tobin, M.J., Jubran, A., Laghi, F.: Patient-ventilator interaction. Am. J. Respir. Crit. Care Med. 163(5), 1059–1063 (2001)
37. Vignaux, L., Vargas, F., Roeseler, J., Tassaux, D., Thille, A., Kossowsky, M.P., Brochard, L., Jolliet, P.: Patient-ventilator asynchrony during non-invasive ventilation for acute respiratory failure: A multicenter study. Intensive Care Med. 35(5), 840–846 (2009)
38. de Wit, M., Miller, K., Green, D., Ostman, H., Gennings, C., Epstein, S.: Ineffective triggering predicts increased duration of mechanical ventilation. Crit. Care Med. 37(10), 2740–2745 (2009)
39. Wrigge, H., Reske, A.: Patient-ventilator asynchrony: Adapt the ventilator, not the patient! Crit. Care Med. 41(9), 2240–2241 (2013)
40. Xiaoqing, J., Donzé, A., Deshmukh, J.V., Seshia, S.A.: Mining Requirements from Closed-loop Control Models. In: Proc. of HSCC 2013, pp. 43–52. ACM (2013)
41. Yang, H., Hoxha, B., Fainekos, G.: Querying Parametric Temporal Logic Properties on Embedded Systems. In: Nielsen, B., Weise, C. (eds.) ICTSS 2012. LNCS, vol. 7641, pp. 136–151. Springer, Heidelberg (2012)
42. Younes, H.L.S., Kwiatkowska, M., Norman, G., Parker, D.: Numerical vs. statistical probabilistic model checking: An empirical study. In: Jensen, K., Podelski, A. (eds.) TACAS 2004. LNCS, vol. 2988, pp. 46–60. Springer, Heidelberg (2004)

Track Introduction: Scientific Workflows

Joost N. Kok[1], Anna-Lena Lamprecht[2], Kenneth J. Turner[3], and Katy Wolstencroft[1]

[1] Leiden Institute of Advanced Computer Science, Leiden University, 2300 RA Leiden, The Netherlands
`joost@liacs.nl, k.j.wolstencroft@liacs.leidenuniv.nl`

[2] Chair for Service and Software Engineering, University of Potsdam, 14482 Potsdam, Germany
`lamprecht@cs.uni-potsdam.de`

[3] Computing Science and Mathematics, University of Stirling, Stirling, FK9 4LA, United Kingdom
`kjt@cs.stir.ac.uk`

In recent years, numerous software systems have been developed specifically for supporting the management of scientific processes and workflows (see, e.g., [1] or [2] for surveys). Research in this comparatively new field is currently evolving in interesting new directions. Already at the ISoLA symposium in 2010 we focused on workflow management for scientific applications in the scope of a symposium track on "Tools in scientific workflow composition" [3]. It comprised papers on 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. This ISoLA 2014 special track on "Scientific workflows" focuses again on the various topics connected to scientific processes and workflows. The track comprises five papers, of which three are concerned with concrete workflow applications, and two with the analysis and annotation of (scientific) workflows. They are surveyed briefly in the following. The papers describe research with four different workflow management systems in a broad range of scientific disciplines, including bioinformatics, biomedical sciences, climate change, and robotics. The diversity of approaches and research domains covered will allow participants to share experiences and explore cross-cutting concerns in workflows research.

Workflow Applications

The paper **Meta-analysis of Disjoint Sets of Attributes in Large Cohort Studies** [4] (by Jonathan Vis and Joost Kok) describes a workflow application from the biomedical field. It introduces the problem of classification in large cohort studies with heterogeneous data and proposes an approach for cross-sectional investigation of the data to see the relative power of the different groups. The authors use the Weka KnowledgeFlow environment [5] to define automated workflows for data cleaning, data selection, classifier training and meta-learning, which can again be combined into larger workflows.

Towards a flexible assessment of climate impacts: The example of agile workflows for the ci:grasp platform [6] (by Samih Alareqi, Steffen

Kriewald, Anna-Lena Lamprecht, Dominik Reusser, Markus Wrobel and Tiziana Margaria) is an example from the geoinformatics (climate impact research) domain. In this project the jABC modeling framework [7] is used to define climate impact analysis workflows that can easily be used and adapted by researchers without programming experience. The application is based on the data and functionality that is available in the ci:grasp platform (Climate Impacts: Global and Regional Adaptation Support Platform, [8]), but can in contrast to the static web platform be tailored to the user's specific data and scenarios.

A workflow example from the robotics domain is presented in **A visual programming approach to beat-driven humanoid robot dancing** [9] (by Vid Podpečan). Using the Choreographe visual programming environment, a proprietary workflow environment for NAO robots [10], and the Aubio [11] audio signal processing tool workflow components and workflows are developed for teaching a humanoid robot to dance to a given song (with respect to the detected beat).

Workflow Analysis and Annotation

In **jABCstats: An Extensible Process Library for the Empirical Analysis of jABC Workflows** [12] (by Alexander Wickert and Anna-Lena Lamprecht) the authors describe how they used the jABC modeling framework [7] to develop a collection of workflows that make it possible to easily perform different empirical analyses of jABC workflows. While further extensions are envisaged, currently the library can be used to assess workflow sizes, service usage and the use of workflow patterns. The paper presents first results from the analysis of jABC workflows from different scientific application domains.

Finally, the paper **Automatic annotation of bioinformatics workflows with biomedical ontologies** [13] (by Beatriz García-Jiménez and Mark D. Wilkinson) is concerned with an approach to automatically annotate (legacy) scientific workflows and their component services with ontology terms. For the case study presented in the paper, the authors use Taverna [14] workflows from the myExperiment [15] workflow repository and different existing ontologies from the bioinformatics and biomedical domains.

References

1. Taylor, I.J., Deelman, E., Gannon, D.B., Shields, M.: Workflows for E-Science: Scientific Workflows for Grids. Springer (2007)
2. Scientific workflow system - Wikipedia, the free encyclopedia (last accessed June 27, 2014)
3. Kok, J.N., Lamprecht, A.-L., Wilkinson, M.D.: Tools in Scientific Workflow Composition. In: Margaria, T., Steffen, B. (eds.) ISoLA 2010, Part I. LNCS, vol. 6415, pp. 258–260. Springer, Heidelberg (2010)
4. Vis, J.K., Kok, J.N.: Meta-Analysis of Disjoint Sets of Attributes in Large Cohort Studies. In: Margaria, T., Steffen, B. (eds.) ISoLA 2014, Part II. LNCS, vol. 8803, pp. 407–419. Springer, Heidelberg (2014)

5. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The weka data mining software: An update. SIGKDD Explor. Newsl. 11(1), 10–18 (2009)
6. Al-Areqi, S., Kriewald, S., Lamprecht, A.-L., Reusser, D., Wrobel, M., Margaria, T.: Towards a flexible assessment of climate impacts: The example of agile workflows for the ci:grasp platform. In: Margaria, T., Steffen, B. (eds.) ISoLA 2014, Part II. LNCS, vol. 8803, pp. 420–435. Springer, Heidelberg (2014)
7. Steffen, B., Margaria, T., Nagel, R., Jörges, S., Kubczak, C.: Model-Driven Development with the jABC. In: Bin, E., Ziv, A., Ur, S. (eds.) HVC 2006. LNCS, vol. 4383, pp. 92–108. Springer, Heidelberg (2007)
8. ci:grasp 2.0: Home (last accessed June 27, 2014)
9. Podpečan, V.: A visual programming approach to beat-driven humanoid robot dancing. In: Margaria, T., Steffen, B. (eds.) ISoLA 2014, Part II. LNCS, vol. 8803, pp. 436–448. Springer, Heidelberg (2014)
10. Aldebaran Robotics - Humanoid robotics & programmable robots (last accessed June 27, 2014)
11. Aubio, a library for audio labeling (last accessed June 27, 2014)
12. Wickert, A., Lamprecht, A.-L.: jABCstats: An Extensible Process Library for the Empirical Analysis of jABC Workflows. In: Margaria, T., Steffen, B. (eds.) ISoLA 2014, Part II. LNCS, vol. 8803, pp. 449–463. Springer, Heidelberg (2014)
13. García-Jiménez, B., Wilkinson, M.D.: Automatic annotation of bioinformatics workflows with biomedical ontologies. In: Margaria, T., Steffen, B. (eds.) ISoLA 2014, Part II. LNCS, vol. 8803, pp. 464–478. Springer, Heidelberg (2014)
14. Wolstencroft, K., Haines, R., Fellows, D., Williams, A., Withers, D., Owen, S., Soiland-Reyes, S., Dunlop, I., Nenadic, A., Fisher, P., Bhagat, J., Belhajjame, K., Bacall, F., Hardisty, A., Nieva de la Hidalga, A., Balcazar Vargas, M.P., Sufi, S., Goble, C.: The taverna workflow suite: Designing and executing workflows of web services on the desktop, web or in the cloud. Nucleic Acids Research 41(W1), W557–W561 (2013)
15. Goble, C.A., Bhagat, J., Aleksejevs, S., Cruickshank, D., Michaelides, D., Newman, D., Borkum, M., Bechhofer, S., Roos, M., Li, P., Roure, D.D.: myExperiment: A repository and social network for the sharing of bioinformatics workflows. Nucleic Acids Research 38(suppl. 2), W677–W682 (2010)

Meta-analysis of Disjoint Sets of Attributes in Large Cohort Studies*

Jonathan K. Vis[1,2] and Joost N. Kok[1,2]

[1] Department of Molecular Epidemiology, Leiden University Medical Center, Leiden, The Netherlands
[2] Leiden Institute of Advanced Computer Science, Leiden University, Leiden, The Netherlands

Abstract. We will introduce the problem of classification in large cohort studies containing heterogeneous data. The data in a cohort study comes in separate groups, which can be turned on or off. Each group consists of data coming from one specific measurement instrument. We provide a "cross-sectional" investigation on this data to see the relative power of the different groups. We also propose a way of improving on the classification performance in individual cohort studies using other cohort studies by using an intuitive workflow approach.

Keywords: meta-analysis, machine learning, data mining, classification, feature selection, cohort studies.

1 Introduction

Cohort studies are frequently used in the biomedical field. The aim is to identify so-called risk factors correlated to a phenotype, usually a disease. A cohort is a group of people sharing a common characteristic during a certain period. Within this period some people either develop the studied phenotype or already exhibit it from the start. This subgroup is referred to as the *case* group, while the remainder of the people are designated as *controls*.

Nowadays a variety of different types of biomedical data can be gathered. Often physiological data such as gender, age, blood pressure and heart rate are present together with genomics data either consisting of the complete genomic sequence, but more often, in the form of *single nucleotide polymorphisms* (SNPs). While some disease have a clear genomic origin, many others are cause by a more complex combination of effects, therefore, data about the presence or concentration of all kinds of substances within the body such as *metabolites*.

As new data generating methods become available during the period of a cohort study data is accumulated. Different types, e.g., genomics or metabolites, of data are collected from the same group of people resulting in more of less disjoint data sets describing a self-contained set of attributes of the same individual. It is unclear whether these data sets have the power to augment each other or

* This publication was supported by the Dutch national program COMMIT.

whether they express the same knowledge. For example, due to a genomic defect a certain substance (metabolite) is under- or overexpressed.

The abundance of attributes per person results often in a skewed data set; a few instances versus a lot of attributes. As a consequence finding correlations in the data becomes tricky This effect is enhanced by the fact that widening (adding attributes) a data set is cheaper then adding more individuals. Which is even impossible at a later stage of the study.

Adding data is never free. This is especially the case in the biomedical field. The preparation and construction of these data sets are labor intensive and expensive in a financial sense. Furthermore, this process imposes a burden on the individuals from whom the samples must be taken. This results in a clear motivation to study the usefulness of the gathering of groups of additional data.

In this paper we investigate three cohort studies which each consists of a number of sets of attributes. In contrast to the more generally applied feature selection, we add whole sets of features instead of single features. We are primarily not so much interested in the absolute classification, but more in the added improvement on the classification of these separate sets. In addition, we propose a method of improving classification in one cohort study by adding data from different cohort studies.

The remainder of the paper is organized as follows. In Section 2 we formalize a problem statement as well as introduce the cohort studies used in the experiments. Section 3 describes the workflow and tooling. We present the experiments in Section 4, and the conclusions to the study in Section 5.

2 Problem Statement

In contrast to the classic feature selection problem, we have a number of disjoint sets of attributes that can either be included as a whole or they can be completely excluded from the study. Furthermore, we have a number of separate studies (concerning the same phenotype and covering the same sets of attributes) that can be pooled together in order to augment classifying power on a separate study. Given this setting, we define two meta-analysis problems:

1. Can we say something about the relative power of the (combinations of) sets of attributes?
2. Can data from different cohort studies be used to augment classifying power for a single study?

2.1 Anatomy of the Data Sets

As we are dealing with humans represented in these cohort studies, we must observe caution not to accidentally expose any details of these individuals nor are we authorized to publish any identifying details about the studies themselves. Therefore we choose not to describe the actual cohort studies used, but we will give a feeling for the characteristics of these studies.

In the cohort studies we are interested in the classification of a certain phenotype: an aging related disease. Earlier attempts to characterize this disease have resulted in a reasonably small (about 10) set of so-called risk factors which seem highly correlated with this disease. These risk factors are all physiological (e.g., age and sex), and, compared to many of the other attributes, easily gathered.

In this study we will use three real cohort studies. The typical number of instances per study is between 1,000 and 2,000 of which approximately 25% is an identified case example (an individual exhibiting the disease).

2.2 Disjoint Sets of Attributes

The data in these cohort studies can be partitioned in disjoint sets of attributes describing a certain type of features. The first set we consider is the set of risk factors derived from earlier studies. We use this as a baseline for our study. Next, we have a set of several dozens of general physiological and behavioral characteristics. Both sets are rather easily constructed and therefore commonly present in similar studies. We have a set of genetic data containing several hundreds of SNPs, which are already selected as promising candidates for correlation regarding this disease. And finally, two sets of metabolitic data: concentration levels of several dozens of Free Fatty Acids (FFA), and about a hundred metabolites measured with NMR (nuclear magnetic resonance) spectroscopy, represented as areas under curve.

As the last three sets of data are considerably more difficult in terms of labor and finances to gather we are especially interested in their "added value" with regard to the general study and each other.

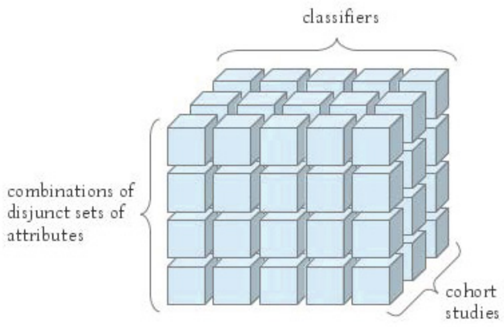


Fig. 1. Schematic representation of the mining space. Each cube represents the classification power for the corresponding classifier on a combination of disjoint sets of attributes from a certain cohort study.

3 Workflows

To find answers to the problems stated in Section 2, we consider all possible combinations of the disjoint sets of attributes (except for selecting no data at all) from all combinations of cohort studies. For each partition of the data we calculate the classification power of all classifiers, see Figure 1.

We propose an automated workflow for the calculation of all data points in Figure 1 in Figure 2. As a first approach all workflows are implemented as batch scripts using the pipes and filters design pattern. In particular a selection of the standard Unix tools is used to perform the splitting of the respective data sets, the creation of folds, and data cleaning. We often augment the pipes and filters pattern [4] by the data-driven pattern of the `make` utility. Commonly `make` is used to automatically build executables from source code, however it is not limited to building software. Indeed its data-driven paradigm combined with the power of declarative programming makes it especially useful in data centric workflows as described in [11].

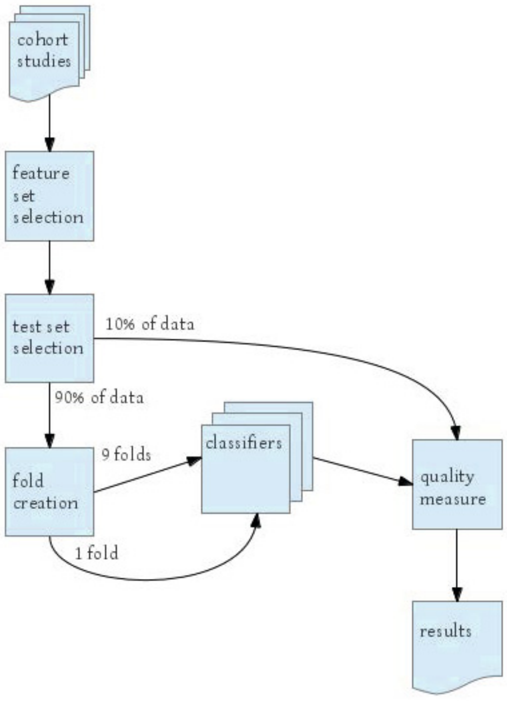


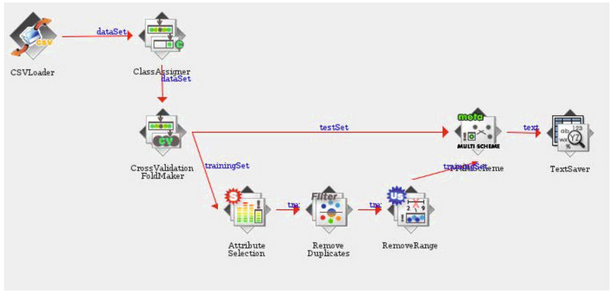
Fig. 2. Graphical representation of the automated mining process. It shows the partitioning of the data and its distribution over the classifiers.

Although workflows have been introduced in the biomedical field, e.g., [9], we observe that *ad hoc* solutions are frequently used preferring agile development over reusability and scalability. In our particular case we want to combine data from separate data sets making the need for scalability more important. Furthermore we acknowledge that our first approach is highly technical and probably difficult to maintain in the biomedical field. Most researchers in this field dealing with the data mining tasks on large cohort studies are not computer scientists or programmers. In order to enable them to conduct their independent research on these data sets, we have to provide them with high-level and easy to understand formal workflows.

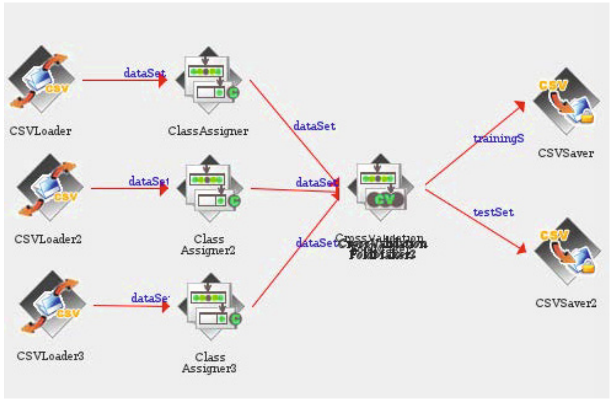
To make our workflow accessible and reusable we introduced a more formal workflow. As most of our classifiers (see Section 3.1) are taken from the Weka toolbox [5], it seems natural to use the Weka KnowledgeFlow environment to design our workflows. Even though this environment is hardly a general workflow tool, our primary tasks are data mining related further advocating the use of this specialized toolkit.

To cater for a number of different recurring subtasks we designed four separate workflows in the Weka KnowledgeFlow environment, see Figure 3:

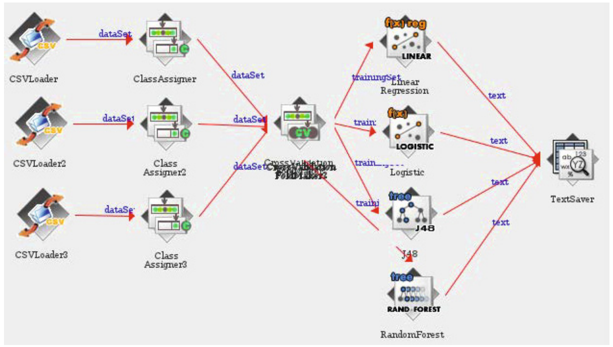
(a) Data cleaning — In order to link several data sets together it is imperative that corrupt and inaccurate instances and attributes are detected and corrected;
(b) Data selection — Usually a single data set is used for training and validation purposes, here we deal with data from separate sets. This workflow enables the preselection of instances from these separate sets;
(c) Classifier training — This is the backbone of the actual data mining process. It is a fairly standard classifier training workflow;
(d) Meta-learning — To effectively combine the results from the individual mining processes, we use the meta-learning workflow. It deals in particular with the hierarchical approach discussed in Section 4.4.



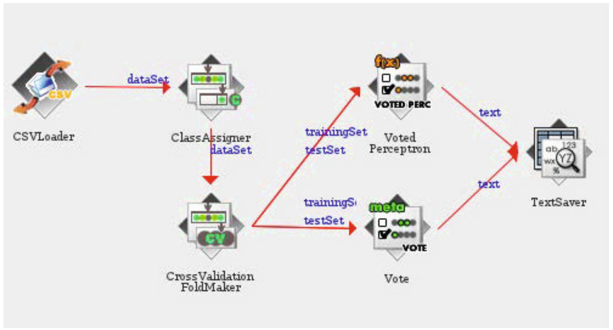
(a) Data cleaning workflow. A simplified version is shown here: not all filters are present.



(b) Data selection workflow.



(c) Classifier training workflow. Not all classifiers are included in this representation.



(d) Meta-learning workflow.

Fig. 3. Weka KnowledgeFlow workflows

These separate workflows can in turn be combined into larger workflows if desired. Note that the meta-learning workflow is similar to the normal learning workflow, making the meta-level concept easier to understand, which is visualized in the actual workflows.

Although the Weka KnowledgeFlow environment might not be a full-scale workflow modelling language, it is well-suited for the analyses in this paper. By using largely classifiers from the Weka toolbox we avoid issues of incorporating different data mining tools. In a more general case it is often advantageous to use many different tools. Under this precondition it becomes difficult to solely use the Weka toolkit as a workflow modelling language and it is recommended to use a general workflow modelling framework like Taverna [15]. For this paper we will consider this to be future work.

3.1 Classifiers

In our case we are interested in classification: to find causal attributes for the presence of a phenotype. The data at our disposal is highly heterogeneous in nature, therefore, we will use a variety of classifiers each especially suited to one or more types of attributes. We include also two techniques from the statistical field for comparison: regression analysis. Statistics are still widely used within the biomedical field.

We acknowledge the fact that there are many more methods available, however, we tried to create a “cross-section” of the many types of classifiers existing today. A clear focus is on the more widely used methods. We present each method together with a motivation why this particular method seems to be useful on our data as well as the tool/implementation used:

- Logistic regression [6] — often used in statistical analysis, here used as comparison;
- Least squares [1] — often used in statistical analysis, here used as comparison;
- Bayesian network [10] — often used in the medical field;
- Decision tree (C4.5) [12] — baseline for many knowledge discovery methods;
- Random forest [2] — baseline for many knowledge discovery methods;
- Neural network (multilayer perceptron with one hidden layer) [13] — especially useful for the large quantities of numerical data;
- Support vector machine (linear) [3] — baseline for many knowledge discovery methods;
- Subgroup discovery [7,8] — identifying subgroups might be useful in describing possibly different forms of the disease.

For practical reasons mostly implementations from the Weka toolbox are used. We believe that the actual implementation of the algorithm is not affecting the results. Nor are we primarily interested in the actual classifying power, but more in the added value of the disjoint sets of attributes.

3.2 Quality Metrics

A lot of different quality metrics have been used to describe the performance of classifiers. Most of them are defined in term of the confusion matrix containing the number of true positives, true negatives, false positives, and false negatives. Commonly used metrics include: precision and recall, sensitivity and specificity, and receiver operating characteristic. As we are trying to characterize the relative performance of several methods, we choose a method that can be expressed as a single number. We used the so-called weighted relative accuracy (WRAcc) [14]:

$$\text{WRAcc}(\text{Class} \leftarrow \text{Cond}) = p(\text{Cond}) \cdot (p(\text{Class}|\text{Cond}) - p(\text{Class})).$$

The WRAcc embodies a trade-off between standard accuracy and generality without sacrificing to much accuracy. Often this metric performs well and tends to yield fewer and simpler patterns, which are considered to be an asset in the biomedical field as usually the resulting patterns have to be explained in this field.

4 Experiments

In this section we will describe the computer experiments. Note that we are not primarily interested in the classification performance of the disease, but rather to investigate the effects of augmenting the separate sets with each other with regard to this classification problem.

In all experiments we use *stratified* 10-fold cross-validation. Each data set is randomly partitioned into ten equal size subsamples in such a way that the proportion of the cases versus the controls is constant. A single subsample is designated to be the validation data for the trained model, while the remaining nine subsamples are combined to train the model.

As a baseline we use the current standard risk prediction method derived from earlier studies, which is a non-linear function over all features captured within the risk factor set. The performance of this method on our three studies is described in Table 1.

Table 1. The performance of the current standard risk prediction method

Study	WRAcc
Cohort Study I	79.08
Cohort Study II	84.90
Cohort Study III	87.13

4.1 Classification Power of Disjoint Sets of Attributes

As a first experiment we investigate the additional classification power gained by adding each disjoint data set. We did this for all three cohort studies. We took all combinations of the disjoint sets and calculated their respective predictive power, see Table 2. We do not show the combinations where the risk factors are included, because they result in every case in a WRAcc that is very close to the WRAcc of the risk factors alone. With the notable exception of the "all" data set, where all attributes (including the risk factors) are considered. Note that the set of FFAs is not available for Cohort Study II.

Table 2. The WRAcc for combinations of the disjoint sets of attributes

Cohort Study I								
set(s)	logistic regression	least squares	bayesian systems	decision tree	random forest	neural network	SVM	subgroup discovery
all	80.98	73.98	59.96	74.63	78.07	58.09	66.59	62.03
risk factors	73.08	64.55	55.26	71.79	71.14	51.22	63.99	57.38
NMR	62.63	57.45	50.08	62.40	65.08	50.00	57.02	56.44
SNP	65.74	60.01	59.00	63.87	61.79	51.52	56.41	56.04
FFA	66.56	56.29	54.14	62.98	64.01	53.74	58.00	57.25
{NMR, SNP, FFA}	70.15	57.78	50.26	63.57	70.60	52.59	61.15	55.80
{NMR, SNP}	65.12	62.59	50.11	65.21	64.09	53.97	62.05	51.09
{NMR, FFA}	68.07	63.67	53.28	62.20	61.52	54.07	59.22	54.31
{SNP, FFA}	65.57	58.73	58.21	58.40	66.93	53.99	58.49	54.73

Cohort Study II								
set(s)	logistic regression	least squares	bayesian systems	decision tree	random forest	neural network	SVM	subgroup discovery
all	90.02	81.05	67.80	86.18	88.42	54.92	85.05	58.51
risk factors	89.98	76.50	74.61	85.98	83.60	52.76	83.73	68.59
NMR	75.61	61.57	56.47	65.99	70.52	57.63	67.72	55.76
SNP	69.61	67.13	60.66	72.61	72.98	57.01	77.53	61.25
{NMR, SNP}	72.70	60.01	57.51	72.93	72.65	59.13	63.17	51.81

Cohort Study III								
set(s)	logistic regression	least squares	bayesian systems	decision tree	random forest	neural network	SVM	subgroup discovery
all	93.97	79.32	77.60	84.72	87.35	70.88	83.85	77.35
risk factors	89.98	72.22	52.07	85.05	77.61	65.65	75.56	67.04
NMR	74.45	68.23	51.72	72.05	82.16	58.47	71.61	50.99
SNP	73.85	61.91	57.59	69.72	70.94	57.81	69.79	53.21
FFA	73.55	57.78	52.76	79.89	76.33	58.66	69.61	67.46
{NMR, SNP, FFA}	77.14	66.07	54.45	70.60	78.21	56.98	70.11	57.00
{NMR, SNP}	75.21	71.43	58.61	73.98	66.89	53.47	75.36	58.25
{NMR, FFA}	77.86	70.45	72.20	78.76	76.51	50.73	69.56	60.79
{SNP, FFA}	76.63	65.84	62.97	65.41	76.33	63.89	66.48	51.66

As can be observed from Table 2, the small set of risk factors is able to outperform all of the other combinations of (much larger) sets of attributes in term of absolute predictive power regardless of the classifier. However, some classifiers are able to outperform the risk factors at specific sets. For instance, neural networks seem to perform better on some of the sets of attributes that contain predominantly numerical data. Furthermore, in all cases providing all data to the classifier improves its prediction power. This is not generally true. As most classifiers use a heuristic method to combine certain attributes, it can be the case that adding data obscures the underlining patterns, which result in a reduced performance. This effect can also be observed in Table 2, e.g., when combining {NMR, SNP, FFA} which results sometimes in a lower WRAcc than the combination {NMR, FFA}.

Based on the results for the three cohort studies, we cannot draw statistically significant conclusions about the relative power of the groups of attributes. For this we should include additional cohort studies. The meta-analysis method can then provide very useful insights for the classification problems of the cohort studies.

4.2 Using Classifiers across Cohort Studies

In this second experiment, we are interested in the performance of the classifiers (trained and validated on their respective cohort studies) in Table 2 on other studies. We hope to acquire insight in the generality of the classifiers. In Table 3, we present the measured predictive power of the classifier trained and validated on Cohort Study I (as this is the largest) tested on Cohort Study II and Cohort Study III. Note that the set of FFAs is not available for Cohort Study II.

Table 3. The performance (WRAcc) of the classifiers trained on Cohort Study I and tested on Cohort Study II and Cohort Study III

Cohort Study II set(s)	logistic regression	least squares	Bayesian systems	decision tree	random forest	neural network	SVM	subgroup discovery
all	75.69	58.03	51.74	60.76	69.65	53.13	56.37	57.93
risk factors	61.08	61.34	52.49	63.13	64.85	49.70	67.97	56.05
NMR	59.90	51.00	50.93	62.85	67.60	52.88	56.62	49.10
SNP	58.35	66.65	59.59	59.47	60.93	53.54	53.12	45.64
{NMR, SNP}	56.67	54.11	36.01	63.43	72.22	48.28	68.04	52.32

Cohort Study III set(s)	logistic regression	least squares	Bayesian systems	decision tree	random forest	neural network	SVM	subgroup discovery
all	54.01	49.98	36.72	53.84	57.42	64.42	48.44	43.47
risk factors	57.77	42.50	28.42	52.18	67.53	55.16	61.57	49.83
NMR	55.31	50.87	32.19	59.29	64.09	46.22	61.01	53.77
SNP	41.52	46.40	42.65	50.71	61.71	59.21	58.38	38.71
FFA	46.46	47.73	53.07	52.13	63.96	59.29	55.50	40.19
{NMR, SNP, FFA}	51.80	55.18	47.43	50.43	47.54	58.22	52.49	44.19
{NMR, SNP}	52.83	50.86	37.63	56.65	69.83	55.98	50.20	34.31
{NMR, FFA}	48.39	39.44	30.73	49.68	65.93	58.51	50.64	49.15
{SNP, FFA}	58.05	44.58	29.26	59.19	55.67	58.19	58.49	50.07

The results in Table 3 are not as good as expected. Although the performance on Cohort Study I is reasonable good, the performance on Cohort Study III is not very good, as, most scores are just above 50%. Apparently the model for Cohort Study I is not capable of predicting the disease in more general cases. The predictive power maintained on Cohort Study I is notably better. A possible explanation being the low number of cases in Cohort Study III. It might even be possible that these cases are of a different type with regard to the predominant cases characterized in Cohort Study I as overfitting seems not to be the problem when training on the Cohort Study I.

The remaining forms of cross model testing are not shown here as they yield significantly less performance results. Again presumably because of the low ratio of cases versus controls in this study.

4.3 Combining All Data from Different Studies

As is apparent from the experiments in Section 4.2, the generality of the constructed classifiers can be improved. We introduce a new experiment: instead of training and validating on a separate study we will combine all data into a single data set. 90% of this data set is used to train and validate the classifiers (as usual using 10-fold cross validation). The resulting classifiers are then tested on the respective 10% of the disjoint sets from their original cohort studies. Thus avoiding the pitfall of overfitting. These results are shown in Table 4.

Table 4. WRAcc of the classifiers trained on all data combined and tested on disjoint sets of attributes

Cohort Study I set(s)	logistic regression	least squares	Bayesian systems	decision tree	random forest	neural network	SVM	subgroup discovery
all	58.23	59.19	51.70	62.63	65.30	69.52	58.04	46.51
risk factors	54.40	58.03	45.04	57.28	57.56	57.10	55.21	40.09
NMR	52.12	42.76	45.95	57.91	59.58	49.18	52.54	40.94
SNP	44.50	43.31	50.86	53.83	50.53	62.42	56.36	36.24
{NMR, SNP}	52.87	40.10	49.21	58.85	48.43	54.47	39.50	43.53
Cohort Study II set(s)	logistic regression	least squares	Bayesian systems	decision tree	random forest	neural network	SVM	subgroup discovery
all	78.60	62.16	62.75	69.94	72.45	59.24	70.33	65.39
risk factors	73.15	53.76	45.12	57.57	60.89	56.31	50.48	58.42
NMR	64.36	59.89	59.15	62.02	67.19	52.38	65.13	55.43
SNP	61.82	62.74	54.49	60.99	66.53	50.89	50.69	56.50
{NMR, SNP}	68.79	54.79	62.04	65.00	64.20	48.63	69.18	50.98
Cohort Study III sets(s)	logistic regression	least squares	Bayesian systems	decision tree	random forest	neural network	SVM	subgroup discovery
all	65.20	59.21	64.09	66.01	70.45	58.88	65.83	62.41
risk factors	59.50	59.41	62.51	55.83	69.86	58.14	54.11	47.66
NMR	60.81	52.77	50.66	53.76	68.82	49.10	56.10	58.86
SNP	55.78	51.16	45.95	63.36	66.35	54.66	45.71	45.90
{NMR, SNP}	60.16	51.57	61.45	59.84	55.08	52.61	65.55	59.26

The predictive power described in Table 4 is indeed more promising than in Table 3. However, we lose a lot of accuracy on all of the separate studies at the benefit of a more stable (general) classifier. We feel that the added benefit of stability does not outweigh the medical consequences of losing that much performance, and is, therefore, not satisfactory in practise.

4.4 A Hierarchical Approach

Next we present a hierarchical approach to improve the overall performance of the classifiers on the disjoint sets of attributes. Based on the observation in Table 2 that some classifiers yield better results on certain sets. We want to make use of this feature by applying the best performing classifier on each of the disjoint sets, and combine their predictions in some way. Note that we will consider the single sets only. Applying this method to all combinations of the disjoint sets will be regarded as future work. Reminding the results from the experiments performed in Section 4.2 and Section 4.1, we will once again consider the studies separately.

We will use two methods of combining the predictions of the individual classifiers: majority voting and a linear perceptron, a simple form of a neural network consisting of only an input layer (the predictions) and an output layer of one node: the ultimate prediction based on the predictions of the individual classifiers. In case of majority voting we do not need to train the hierarchical method, so no addition data is required. This is not the true for in case of the linear perceptron that needs to be trained as well. We use the same partitioning strategy as used in Section 4.3 in order to avoid overfitting. In Table 5 we present the performance of the hierarchically linked classifiers trained on their respective studies, but selected based on their individual performance on Cohort Study III, and, in case of the classifiers trained on the Cohort Study III data, the data from Cohort Study I.

Table 5. The WRAcc of the hierarchically linked classifiers

study	majority	linear perceptron
Cohort Study I	73.43	82.83
Cohort Study II	83.28	90.74
Cohort Study III	92.33	94.55

As is apparent in Table 5, we are able to improve upon the predictive power of the individual classifiers as is shown in Table 2 when using a linear perceptron. The majority voting approach seems to be inferior to using the flat data. Probably, this is caused by the fact that all data sets are weighted equally, where as analyzing the data directly retains the possibility of weighting groups and individual attributes differently. The (small) increase in performance of the hierarchical approach by using a linear perceptron might be explained by a divide-and-conquer argument. By presenting the classifiers with smaller data set they are more easily capable of fitting a model there on. Apparently, these individual models can be combined just by using a simple weighting scheme.

5 Conclusions

In this paper we investigated the added benefits of augmenting large cohort studies with disjoint sets of attributes and data from other studies. We provided a "cross-sectional" study of different classifiers on the heterogeneous data available within these studies. As expected, knowledge discovery on this kind of studies is not a trivial task. We compared our results with a baseline standard risk prediction method used in literature. We shown that our methods are able to match its performance, and, in some cases, are able to outperform it. In particular an hierarchical approach seem to yield good results. We have demonstrated that applying machine learning techniques combined with workflow tooling are valuable in solving this task.

As this is rather a small preliminary study, extending this research onto more large cohort studies, and investigating for different phenotypes is regarded to be a valid pointer towards future research.

References

1. Borowiak, D.: Linear Models, Least Squares and Alternatives. Technometrics 43(1), 99 (2001)
2. Breiman, L.: Random Forests. Machine Learning 45(1), 5–32 (2001)
3. Cristianini, N., Shawe-Taylor, J.: An Introduction to Support Vector Machines and Other Kernel-based Learning Methods. Cambridge University Press (2000)
4. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-oriented Software. Addison-Wesley Longman Publishing Co., Inc., Boston (1995)
5. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The weka data mining software: An update. SIGKDD Explor. Newsl. 11(1), 10–18 (2009)
6. Hosmer Jr., D.W., Lemeshow, S., Sturdivant, R.X.: Applied Logistic Regression. Wiley (2013)
7. Lavrač, N., Kavšek, B., Flach, P., Todorovski, L.: Subgroup discovery with CN2-SD. The Journal of Machine Learning Research 5, 153–188 (2004)
8. Meeng, M., Knobbe, A.: Flexible Enrichment with Cortana Software Demo. In: Proceedings of BeneLearn, pp. 117–119 (2011)
9. Oinn, T., Addis, M., Ferris, J., Marvin, D., Senger, M., Greenwood, M., Carver, T., Glover, K., Pocock, M.R., Wipat, A., Li, P.: Taverna: A Tool for the Composition and Enactment of Bioinformatics Workflows. Bioinformatics 20(17), 3045–3054 (2004)
10. Pearl, J.: Causality: Models, Reasoning and Inference, vol. 29. Cambridge University Press (2000)
11. Robinson, C., Thain, D.: Automated packaging of bioinformatics workflows for portability and durability using makeflow. In: Proceedings of the 8th Workshop on Workflows in Support of Large-Scale Science, WORKS 2013, pp. 98–105. ACM, New York (2013)
12. Rokach, L.: Data Mining with Decision Trees: Theory and Applications, vol. 69. World Scientific (2007)
13. Rosenblatt, F.: Principles of Neurodynamics. Perceptrons and the Theory of Brain Mechanisms. Technical report, DTIC Document (1961)
14. Todorovski, L., Flach, P., Lavrač, N.: Predictive Performance of Weighted Relative Accuracy. In: Zighed, D.A., Komorowski, J., Żytkow, J.M. (eds.) PKDD 2000. LNCS (LNAI), vol. 1910, pp. 255–264. Springer, Heidelberg (2000)
15. Wolstencroft, K., Haines, R., Fellows, D., Williams, A., Withers, D., Owen, S., Soiland-Reyes, S., Dunlop, I., Nenadic, A., Fisher, P., et al.: The Taverna workflow suite: Designing and executing workflows of Web Services on the desktop, web or in the cloud. Nucleic Acids Research 41(W1), W557–W561 (2013)

Towards a Flexible Assessment of Climate Impacts: The Example of Agile Workflows for the ci:grasp Platform

Samih Al-Areqi[1], Steffen Kriewald[2], Anna-Lena Lamprecht[1], Dominik Reusser[2], Markus Wrobel[2], and Tiziana Margaria[1]

[1] Chair for Service and Software Engineering, Potsdam University, 14482 Potsdam, Germany
http://www.cs.uni-potsdam.de/sse
{samih,lamprecht,margaria}@cs.uni-potsdam.de

[2] Potsdam Institute for Climate Impact Research (PIK) 14412 Potsdam, Germany
http://www.pik-potsdam.de
{kriewald,reusser,wrobel}@pik-potsdam.de

Abstract. The Climate Impacts: Global and Regional Adaptation Support Platform (ci:grasp) is a web-based climate information service for exploring climate change related information in its geographical context. We have used the jABC workflow modeling and execution framework to make flexibilized versions of the processes implemented in ci:grasp available to the scientific community. The jABC permits us to leverage the processes to an easily accessible conceptual level, which enables users to flexibly define and adapt workflows according to their specific needs. The workflows are suitable as graphical documentation of the processes and are directly repeatable and reusable, which facilitates reproducibility of results and eventually increases the productivity of researchers working on climate impact risk assessment. In this paper, we use variations of workflows for the assessment of the impacts of sea-level rise to demonstrate the flexibility we gained by following this approach.

Keywords: scientific workflows, agile methods, model-driven development, climate information, climate impact risk assessment.

1 Introduction

Analyzing and assessing potential impacts of climate change are critical and challenging tasks that require the processing of large and heterogeneous datasets. These analyses are particularly demanding because of the multi-scale and multi-objective nature of environmental modeling for climate change impact assessment [16,9]. Harmonization efforts of scenarios and input data for impact modeling allow for the first time to compare results and assess major sources of uncertainty [31]. There is a large number of emission scenarios, climate models, impact models and fundamental modeling assumptions available. Combining different modeling approaches, various data sources, multiple objectives and basic assumptions for climate impact assessment is a complex challenge.

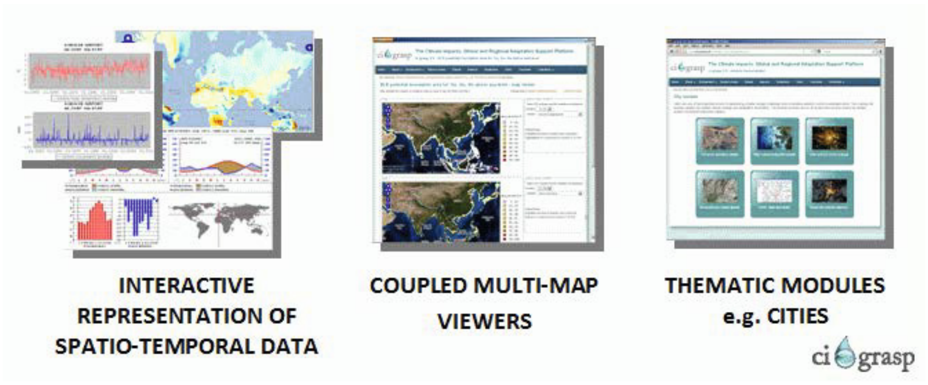

INTERACTIVE REPRESENTATION OF SPATIO-TEMPORAL DATA
COUPLED MULTI-MAP VIEWERS
THEMATIC MODULES e.g. CITIES
ci grasp

Fig. 1. Overview of the information available on ci:grasp

ci:grasp[1] is a web-based climate information service. It aims to support decision makers to better understand impacts of climate change, to prioritize adaptation needs, and to plan and implement appropriate adaptation measures [33,34]. It models climate information and related knowledge in the form of text, maps, and graphs. Main drivers for climate change included on ci:grasp are sea level rise (SLR), change in temperature and precipitation and increased drought risk. These drivers, as well as more frequent extreme events expected under a changing climate, provide a high risk for human lives and cause massive economic damages. Figure 1 gives a short overview of the information presented on ci:grasp.

The ci:grasp platform provides a very rich and well-organized collection of data including 400 adaptation projects and thousands of maps on climate impacts, but it is nonetheless static in the sense that accessible content is being prepared, computed and stored in advance: the processes generating the data displayed on ci:grasp are hidden from the end user and typically carried out by experts calling a number of scripts manually. Our goal is hence to automate these processes and make them visible and accessible for the researchers working with them, so that they can easily and dynamically adapt them to their specific needs and perform analyses tailored to their specific needs.

To this aim, we use the jABC process modeling and execution framework [27] as workflow management system. The jABC is the current reference implementation of the XMDD (eXtreme Model-Driven Design) paradigm [12,23], an extreme version of model-driven development that supports a very flexible and agile development of service-oriented applications by turning system development into user-centric orchestration of intuitive service functionality [22].

[1] `http://www.cigrasp.org`

In this paper, we use variations of workflows for the assessment of SLR related impacts as examples to demonstrate the flexibility that is gained by using the jABC for the management of ci:grasp's processes. The paper is an extended version of a short technical paper [1] accompanying a presentation at this year's International Congress on Environmental Modelling and Software (iEMSs 2014). It is structured as follows. Section 2 introduces the jABC framework, then Section 3 gives some background information on the SLR application example. Section 4 describes how we decomposed the ci:grasp's original assessment of the impact of sea level rise into basic computational services, which we then used as workflow building blocks together with additional functionality from the jABC. Section 5 discusses three examples of workflow variations that we built for the SLR analysis case. Section 6 concludes the paper with a summary, discussion and ideas for future work.

2 The jABC Modeling Framework

The jABC [27] is a multi-purpose and domain-independent framework for model-driven application development. It inherits the power of eXtreme Model-Driven Design (XMDD) [12,23] to enable end users to easily use and compose services into agile workflows. Its way of handling the collaborative design of complex software systems has proven to be effective and adequate for the cooperation of non-programmers and technical people. In fact, the jABC also complies with the best practices for scientific software development recently described by [32] and has been used in a number of scientific workflow projects in the last years (cf., e.g., [21,14,5]). The framework has furthermore been extended by functionality for semantics-based semi-automatic service composition [15,24], which has been shown to be beneficial especially for dealing with variant-rich scientific workflows [13].

The jABC has a comprehensive and intuitive graphical user interface that facilitates workflow development. jABC users easily develop workflow applications by composing reusable building-blocks (called Service-Independent Building Blocks, or SIBs) into hierarchical (flow-) graph structures (called Service Logic Graphs, or SLGs) that are executable models of the application. The workflow development process is furthermore supported by an extensible set of plugins providing additional functionalities, so that the SLGs can be analyzed, verified, executed, and compiled directly in the jABC. Figure 2 gives an impression of the graphical user interface of the jABC in action: The SLG on the canvas has been created using SIBs from the library (displayed in the upper left of the window) in a drag&drop fashion, and connecting them with labeled branches representing the flow of control. After the parameters of the SIBs have been configured (in the SIB inspector in the lower left), the workflow is ready for execution. The small window in the upper left corner of the figure is the control panel of the Tracer plugin that steers the execution of the models. It indicates that it is currently executing a SIB, and the green-colored branches of the model on the canvas visualize where the execution has currently arrived.

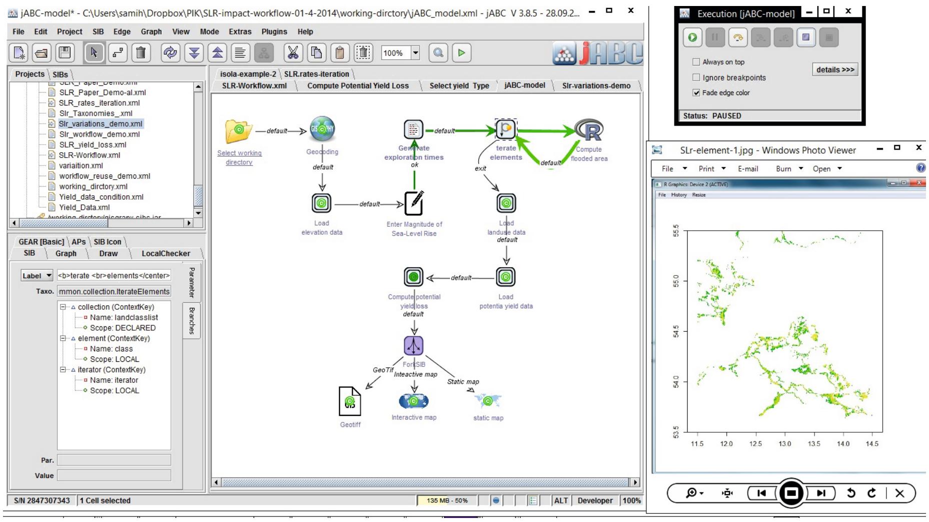


Fig. 2. User interface of the jABC in action

The third window in the figure shows an (intermediate) result from the workflow execution and has been opened by the currently executed SIB.

In contrast to many other scientific workflow management systems (like, e.g., Kepler [17], Triana [28], or VisTrails [3]), whose models represent the flow of data, jABC workflows are control-flow models of the application. While data-flow modeling often appears to be more intuitive in the beginning, it soon reaches its limits when it comes to expressing more complex workflow structures. Control structures like conditional branchings or loops, which are required for supporting the explorative nature of scientific computations, can not be represented by pure data flow models. Therefore, many data flow-oriented systems like those mentioned above include possibilities for defining control flow structures within their workflows. These definitions are however usually less intuitive, and also with them it can still be difficult or impossible to realize particular complex workflow structures that are natural in the control flow formalism (cf., e.g., [13]).

3 Example: Assessing the Impact of Sea-Level Rise

For the example of sea-level rise (SLR) that we focus on in this paper, climate change is assessed with respect to the potential loss of agricultural production, calories available and effect for food security [25], but also with respect to properties of rural and urban damage functions [2]. To this end, heterogeneous data (such as, e.g., elevation, land-use, population density or yield data) has to be used, which comes in different formats and at different scales, requiring adequate integration and aggregation. For the efficient identification of potentially flooded areas, the srtmtools-package [11] for the data analysis language R [26] provides necessary methods to produce results presented on ci:grasp. ci:grasp then allows users to adjust parameters and explore results for coastal regions over the world in an interactive viewer and locate potential need for prevention measures.

For our SLR impact analyses the first step is the identification of potentially vulnerable areas. To this end a digital elevation model (DEM) can be used to identify land-area which is hydraulically connected to the sea and below a certain elevation. This was done following the 8 neighbour rule algorithm implemented in the srtmtools-package. For the included examples the SRTM90 [10] database of terrain elevation with a 90x90 meters spatial resolution and a vertical resolution of 1 meter were used as default dataset. For a given region the srtmtools-package downloads automatically the necessary data from CGIAR-CSI Database [10]. However, any other DEM can also be used.

The second step in SLR impact analysis is the combination of the vulnerable areas with data of interest, for example the type of flooded land[2] or the potential yield- and calorie-loss.

For the evaluation of the potential yields from the threatened crop lands the datasets from the Global Agro-Ecological Zones (GAEZ) [8] were used. The GAEZ provides global information of actual and potential agricultural production, with regard to climate, soil and terrain conditions, such as land cover, irrigation potentials, protected areas, population density, livestock density and accessibility. It describes biophysical limitations and potentials for crop production as well as dependencies on different input levels of fertilizers or different farming techniques. This information is available for a range of different climate scenarios, including a reference climate (average time period 1961–1990). Furthermore, GAEZ is based on published outputs for several IPCC scenario from various global and regional climate models, such as the MPI ECHAM4 model. Overall there are information on yield constraints, crop calendars, harvested area, and production potential estimates for eleven major crop groups, 49 major crops and 92 crop types. Productivity estimates are made for rain-fed farming and several irrigation systems [29].

To deal with different resolutions of various datasets the srtmtools-package provides a re-sampling function which gives the exact amount of flooded land per grid-cell for a different resolution. As a consequence every data which is a linear function of the area can be treated in a similar manner.

4 Domain Modeling

Working with the jABC consists of basically two phases: *domain modeling* and *workflow design*. The domain modeling, as detailed for our example in this section, involves the integration of the required computational services and their organization in domain-specific taxonomies. In particular, we explain how we turned the original srtmtools package into a collection of reusable services that

[2] The MODIS dMCD12Q1 product version 5.1 2010 was used for the land-cover information. Downloaded from the data pool `ftp://e4ftl01.cr.usgs.gov/MOTA/MCD12Q1.051/`

can be used as workflow building blocks in the jABC, and how these services are organized by a newly defined domain-specific service taxonomy.

We use the term *servification* to refer to the process of turning arbitrary software components into proper services that are adequate, for example, for (re-) use in workflow management systems. This is in accordance with the service orientation paradigm, which postulates that any kind of computational resource should be seen and handled as a service – that is, a well-defined unit of functionality with a well-defined interface – to provide a high level of abstraction and reusability (cf., e.g., [19]).

In the Java-based jABC framework, we can in fact use everything as a service that is in some way programmatically accessible. This is apparently the case for Java APIs and Web Services, but includes also classic command line tools, scripts and "headless" operations modes as provided by some desktop applications, which can be executed just based on input parameters and without any user interaction. Depending on the technicalities of the chosen service, its integration into the jABC can be straightforward or challenging, but being able to use the services from an intuitive graphical interface typically outweighs the service integration costs. For more elaborate discussions of costs and benefits of servification in the jABC context, the reader is referred to [4,20,13].

For the servification of the SLR impacts assessment tools described in Section 3, the various existing R scripts, which have been used to produce the data available on the ci:grasp platform, have been decomposed into separate and independent functions and equipped with well-defined inputs and outputs in order to provide proper services adequate for the envisaged application. Technically, these services are currently simply provided in the form of autonomously running R scripts, so that they can easily be encapsulated into SIBs and be used as workflow building blocks within the jABC. So far, 22 services for different data loading, resampling, computations, and data output tasks have been created (see Table 1). To create full workflows in the jABC, these services can be combined with the standard SIB libraries provided by the jABC, for, e.g., data input/output, for evaluating conditions and for basic user interaction.

Figure 3 shows how the SLR services can be taxonomically classified into different groups. Concretely, it defines four subclasses of SLR services: data loading (comprising 5 of the services), resampling (4), computation (8), and output (5). Output contains another subclass to group different services that produce static maps, which comprises two of the output services, while the other three are directly classified as outputs.

5 Workflow Examples

Based on the newly created domain-specific services and the large library of SIBs for common functionality that comes with the jABC framework, we could easily construct different workflows for SLR impact assessment in an agile workflow-based way. The complete project currently comprises around 31 different models, composed of more than 180 SIBs and spanning three hierarchy levels.

Table 1. Services for SLR analysis workflows

Name	Description	Inputs	Outputs
Load SRTM elevation data	Download the digital elevation model (DEM)for the selected area.	Region coordinates	Region.rds file
Compute flooded Areas	Compute the flooded areas for a region based on its DEM.	Region.rds	Landloss.rds file
Load population data	Load population data from global map data based on land loss data.	Population data(.tif) and Landloss.rds	Population.rds file
Load landuse data	Load landuse data from global map data based on landloss data.	Landuse data (.tif) and Landloss.rds	Landuse.rds file
Load yield data	Load yield data (actual or potential).	Yield data (.asc)	Yield.rds file
Load calories data	Load calories data (actual or potential).	Calories data.csv	Calories.rds file
Resample landuse data	Resample land use data with land loss data (flooded areas).	Landloss and Landuse rds files	Landuse-sample.rds files
Resample population data	Resample population data with land loss data (flooded areas).	Landloss and Population.rds files	Population-sample.rds
Resample yield data	Resample yield data with land use data and yield data with landuse-sample and landuse data.	Landuse, Landuse-sample and Yield rds files	Yield-sample and Yield-flooded-sample.rds files
Resample calories data	Resample calories data with land use data.	Landuse, Landuse-sample and Calories.rds	Calories-sample.rds file
Compute population at risk of migration	Estimates the number of people that would be affected.	Population and Population-sample.rds	Result.rds
Compute rural and urban GDP at risk	Estimates potential economic damage in coastal communities.	Population and Population-sample.rds	Result.rds
Compute potential landloss (ha)	Estimates the area that will be potentially inundated.	Landloss and Landuse-sample rds files	Result.rds
Compute land loss classes	Define the type of land affected, from 1 – 16 different land types.	Landuse-sample rds file and number of classes	Several results.rds files
Compute yield loss	Compute actual and potential production value affected in USD.	Yield, Yield-sample and Yield-flooded-sample.rds files	Result.rds
Compute caloric energy loss	Estimates actual and potential number of peoples' annual diets lost.	Calories and Calories-sample rds files	Result.rds
Compute production affected()	Estimates the economic value of the agricultural loss.	Yield, Yield-sample and Yield-flooded-sample.rds files	Result.rds
Generate Interactive map output	Generate an interactive map output using the Google Maps API[5].	Result.rds	Interactive map
Generate Static map Png output	Create static map in Png format.	Result.rds	Png file
Generate Static map Pdf output	Create static map in Pdf format.	Result.rds	Pdf file
Produce GeoTIFF output	Create a geo-referenced file (GeoTIFF, ASCII) which can be used for further external GIS processing.	Result.rds	Geo-referenced file
Produce text output	Create a text file containing some summary and statistic information.	Result.rds	Text file

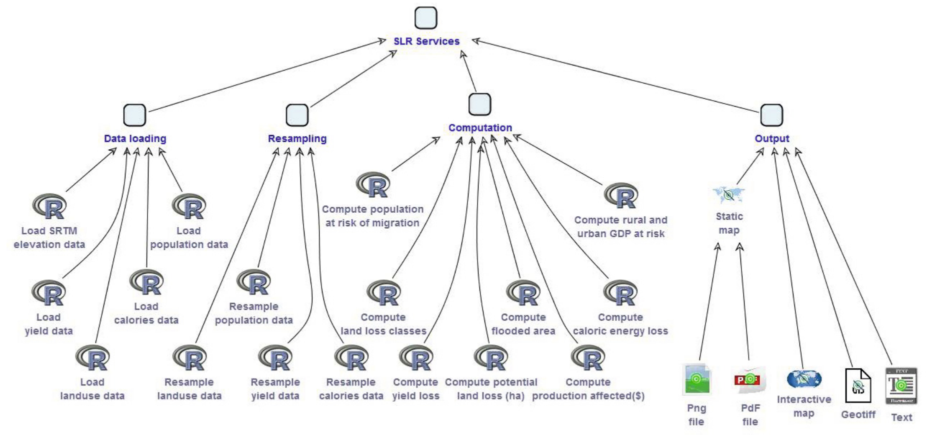


Fig. 3. Taxonomic classification of SLR services

In the following we discuss three selected examples in greater detail: a simple example of an SLR impact assessment workflow as a basis for developing variations (Section 5.1), an extended version of the basic workflow with several predefined variation points (Section 5.2), and a parameter exploration variant that performs the same analysis for a range of SLR values to create data for comparing different possible scenarios (Section 5.3). After that, Section 5.4 sketches further possible variations.

5.1 Basic Workflow

Figure 4 (center) shows a simple workflow for assessing the impact of sea-level rise on the agricultural yield loss for a region to be selected by the user. From left to right (the SIB with the underlined name denotes the starting point), it performs (1) selection of the working directory for input and output data; (2) definition of the investigated area by coordinates; (3) downloading the digital elevation model of the selected area; (4) entering of the magnitude of sea level rise; (5) computation of the flooded area; (6) computation of the yield loss due to the flooding; and (7) generation of an output file with results in PDF format.

Some of the SIBs are marked by a green circle, which indicates that the functionality represented by this building block is actually more complex and defined in a separate model, as so-called submodel. For example, SIB (3) encapsulates a submodel for the selection and loading of elevation data (shown at the top of the figure): A dialog is displayed where the user has to select if he/she wants to use own elevation data or predefined SRTM elevation data with a 90m resolution. Depending on the selection, either the own data is loaded from a file or the predefined data is downloaded from an online source.

As another example for hierarchical modeling, the SIB (6) is a composite service that allows for the computation of several types of yield loss for different

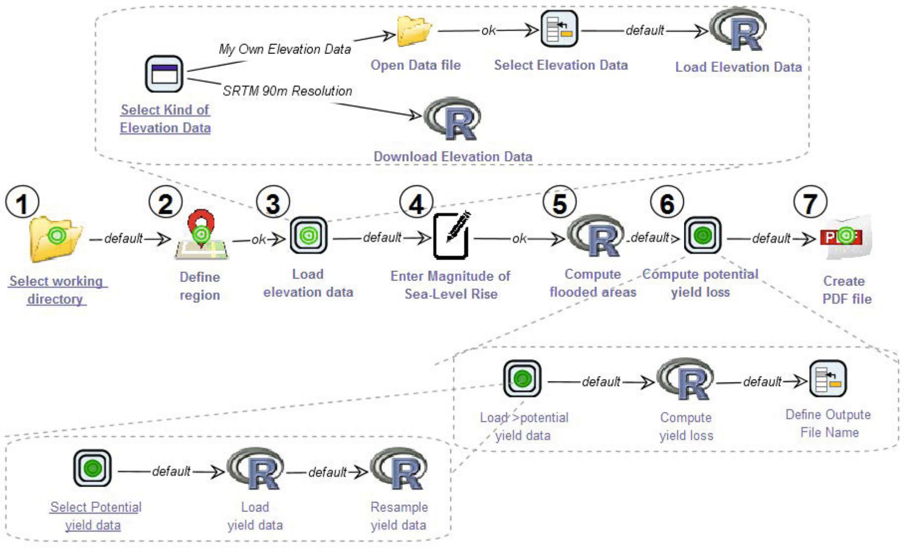


Fig. 4. Basic workflow for SLR impact assessment regarding yield loss

climate scenarios. As the figure (bottom) shows, the submodel contains another submodel, which again contains a submodel. The load potential yield data SIB in the submodel of SIB (6) is in fact another submodel that loads potential yield data, which again makes use of a submodel that handles yield data sets selection. Not shown in the figure, it offers the users the alternatives to work with own data or to select and extract available GAEZ data automatically, similar to the elevation data selection described above.

This hierarchical modeling style allows to organize workflow applications in different levels of abstraction, from coarse-granular and more conceptual views at the higher levels, down to fine-granular and more technical views at the lower levels.

5.2 Workflow with Variation Points

Figure 5 shows an extension of the workflow for SLR impact assessment described above. It comprises a number of preconfigured variation points, which make it easy for the user to build variants of the workflow: he/she just needs to change the execution path in order to include additional options from the variation points, simply by dragging the connecting branches to other SIBs.

Variation point 1 allows for easy modification of the definition of the region considered for the analysis. Besides entering the coordinates directly (as in the basic example), it is also possible to enter the name of a place or an address that is then used as the center of the region, or to select a region interactively on a map.

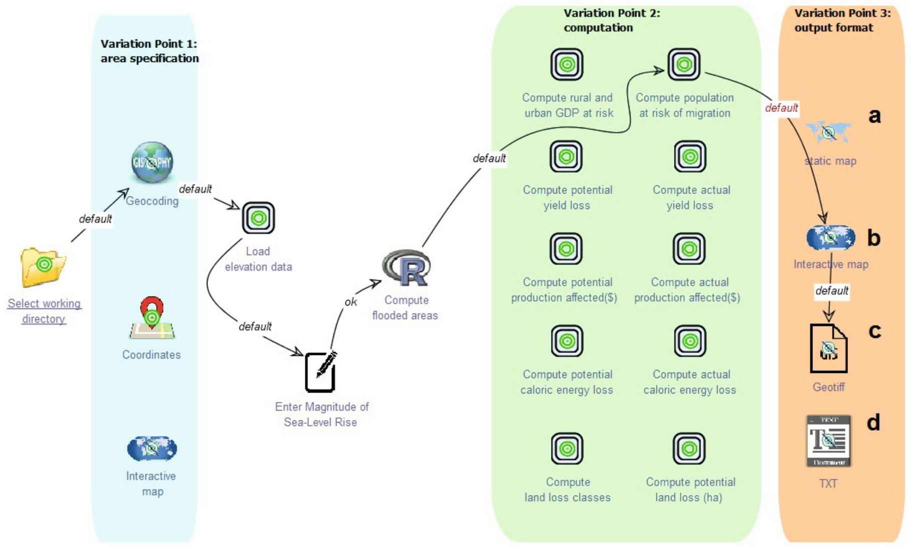

Variation Point 1: area specification
Variation Point 2: computation
Variation Point 3: output format
Select working directory
default
Geocoding
default
Load elevation data
default
Enter Magnitude of Sea-Level Rise
ok
Compute flooded areas
default
Coordinates
Interactive map
Compute rural and urban GDP at risk
Compute population at risk of migration
Compute potential yield loss
Compute actual yield loss
Compute potential production affected($)
Compute actual production affected($)
Compute potential caloric energy loss
Compute actual caloric energy loss
Compute land loss classes
Compute potential land loss (ha)
default
a
Static map
b
Interactive map
default
c
Geotiff
d
TXT

Fig. 5. Workflow for SLR impact assessment with preconfigured variation points

Variation Point 2 contains a collection of different computations, summarized in Table 2, that can be selected according to the concrete objectives when assessing SLR impacts. In fact, some of these computations are using the same concrete computing service. However they behave in different manner based on the variation of data loading services. For example, to compute the yield loss actually two computational services are developed (compute potential and actual yield loss) with respect for analysis objective and for the type of data. As has been sketched in Figure 4, all these computations are in fact realized by separate workflow models.

Variation Point 3 provides SIBs for creating different output formats. Users can select one or more (by including a sequence of output-generating SIBs) formats for the presentation of the final results, including (a) static maps in different formats (jpeg,pdf,png,ps), (b) an interactive map using the Google Maps API[3], (c) a text-file containing a summary and (d) a geo-referenced file (GeoTiff, ASCII) which can be used for further external GIS processing.

Through these variation points, flexible adjustment of the SLR impact assessment is available at the user level. For example, the connections in the Figure 5 may have been defined by user aiming to assess the number of people that potentially affected by a submergence of land triggered by a certain magnitude of sea-level rise: The region is specified via entering the address or name of a place, and taking the surrounding region. After the loading of the elevation data, entering the magnitude of SLR to be considered and computation of the flooded

[3] https://developers.google.com/maps/

Table 2. Overview of computation services of Variation Point 2

SIB	Description
compute rural and urban GDP at risk	focuses on potential economic damage in coastal communities
compute population at risk of migration	focuses on the number of people that would be affected
compute actual/potential yield loss	compute actual/potential production value affected in USD
compute actual/potential production affected ($)	focuses on the economic value of the agricultural loss
compute actual/potential caloric energy loss	focuses on the actual and potential peoples' annual diets lost
compute land loss classes	determine 1 – 16 different land types
compute potential land loss (ha)	determine the area that will be potentially inundated

areas, the computation of population potentially affected is performed. Finally, the outputs are shown in two different views.

5.3 Parameter Exploration Workflow

Generating and comparing projections for different scenarios is a common approach to address the inherent uncertainties in assessing future climate change. Figure 6 shows an example of a workflow that performs a parameter exploration by iterating over different magnitudes of SLR. Here, the user enters not only a single SLR magnitude, but a comma-separated list of potential increases in sea-level. This step of the workflow execution is shown in the Figure 6. The workflow iterates over the list elements, computing the potentially flooded area for a different magnitude of SLR in every iteration. As shown in Figure 7, the user obtains a set of maps, each representing a different scenario. Although this is only a simple example, it demonstrates well the possibility to use the jABC for performing an analysis for parameters automatically, which in this case pro-

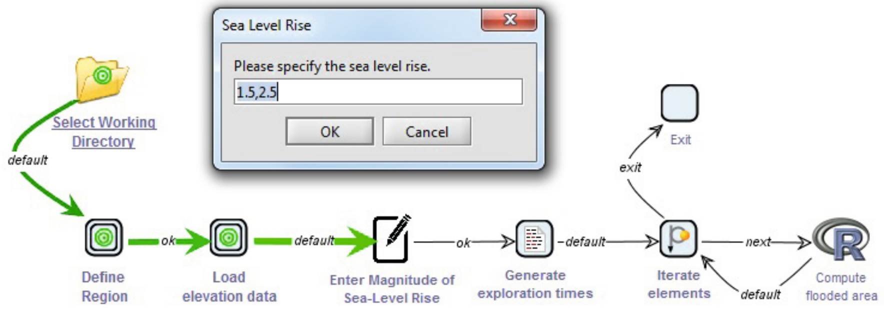


Fig. 6. Workflow for the iteration over different values of SLR

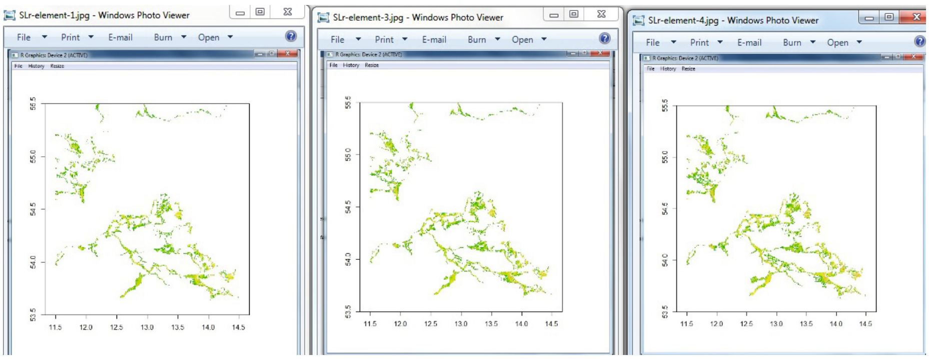

SLr-element-1.jpg - Windows Photo Viewer
SLr-element-3.jpg - Windows Photo Viewer
SLr-element-4.jpg - Windows Photo Viewer
File Print E-mail Burn Open
R Graphics: Device 2 (ACTIVE)
53.5 54.0 54.5 55.0 55.5
11.5 12.0 12.5 13.0 13.5 14.0 14.5

Fig. 7. Exploration of flooded areas with different SLR values (2, 3, 4)

vides a convenient way to generate outputs for multiple scenarios to deal with the uncertainty about the magnitude of SLR under climate change.

5.4 Further Variations

The three examples have already given a good impression of the workflow modeling capabilities of the jABC framework: the SLGs are hierarchical and support reuse of submodels, and being control-flow models of the developed applications they also allow to include essential control structures such as conditional branchings and loops into the workflows. Parallel execution is also supported by a fork/join mechanism that distributes the execution flow into different threads.

With these capabilities in mind, we can imagine, for instance, also the following variations of the SLR impact assessment workflow:

– Exploration of other parameters than the magnitude of SLR, for example different land use data sets and different land loss classes.
– Flexible inclusion of different data. This is easiest if data is available in machine-readable format through the web.
– A combination of an iterative execution with predefined variation points, to make it easier to vary the parameter exploration workflow.

6 Discussion and Conclusion

We showed how the XMDD-oriented workflow development style supported by the jABC process modeling and execution framework permits us to leverage the processes implemented in the ci:grasp platform to a user-accessible level, and thus to enable users to flexibly define and perform multi-objective workflows tailored to their specific needs. The described level of flexibility is essentially achieved by:

– rigorous *service orientation*, turning basic components as well as their compositions into flexibly reusable pieces of functionality,

- *hierarchical modeling* that allows to represent processes on different levels of abstraction (ranging from completely user-accessible top-level workflows that hide all technical details to fine-granular service compositions on the lower levels) and makes them reusable, and
- an *intuitive graphical interface* that makes it easy also for non-programmers to design and adapt workflows according to their specific preferences and constraints.

For the exemplary scenario of assessing the impact of sea-level rise, we discussed how we decomposed the ci:grasp's original implementation into basic computational services, which we then used as workflow building blocks. Together with additional functionality from the jABC, we built a flexible library of directly executable workflows for this type of analysis. With this library of workflows various kinds of SLR analyses can be easily and flexibly created according to the user's objectives and preferences. This is a relevant contribution to the community of climate impact assessment, as the variety of SLR assessment tools is still limited [7]. To illustrate this, we discussed three exemplary workflow incarnations and sketched further possible variations.

Instead of restricting access to a pre-selected set of results these three workflows allow the user to generate tailor-made analyses. To provide worldwide coverage on ci:grasp we made use of datasets available globally. For regional analyses more detailed or accurate data may be available. The basic workflow allows the user to use such regional data for investigating the impact of sea-level rise. In addition, modifying the scenarios through the possibility to choose a user-defined magnitude of sea-level rise may reveal additional insight to the standard scenarios of 1, 2 and 3 m present in ci:grasp, for example to evaluate extreme storm surges. The Web Mapping Service (WMS) technology used in ci:graps does not provide a powerful interface for further detailed analysis. The workflow with Variation Points provides flexibility in the choice of the output format and enables the user to perform unexpected additional analyses which are currently not provided through the web-page.

In addition to the personalized results, such a user friendly and flexible environment for SLR analysis has the potential to motivate sharing and reuse of code. This directly increases the reproducibility of scientific analyses, which is core to the scientific process (cf., e.g., [18]). The sharing of models can be further leveraged by making software publishable, citable and recognized as scientific achievement. A recent movement pursuing these goals is sciforge [6], which aims at establishing the missing link between papers and published data.

Ultimately, we can envision a transformation of environmental impact assessment with easy inclusion of large and heterogeneous data sets, shared and flexible assessment models and powerful visualization of the large set of results [30]. User friendly workflow systems are one essential building block to allow for such a transformation.

Having such a vision in mind, in a next step other impact assessment models than the one applied in ci:grasp may go through a similar servification process, extending the library with additional and alternative services. Note that the response time of computationally expensive models can pose additional challenges in this context. Moreover, more flexible inclusion of various, heterogeneous data sources can be achieved with additional SIBs.

Furthermore, it will be beneficial to improve the multi-objective and multi-scale risk assessment by the use of semantics-based workflow design methodology, similar to the work described by [13]. As the successful application of such methodologies crucially depends on adequate domain modeling, a major part of our future work will focus on the application and design of domain-specific ontologies. Once a semantics-based workflow design framework is available, the integration of the computational part of an impact assessment into a broader adaptation cycle will be possible.

References

1. Al-areqi, S., Kriewald, S., Lamprecht, A.L., Reusser, D., Wrobel, M., Margaria, T.: Agile Workflows for Climate Impact Risk Assessment based on the ci:grasp Platform and the jABC Modeling Framework. In: International Environmental Modelling and Software Society (iEMSs) 7th Intl. Congress on Env. Modelling and Software (accepted, 2014)
2. Boettle, M., Rybski, D., Kropp, J.P.: How changing sea level extremes and protection measures alter coastal flood damages. Water Resour. Res. 49(3), 1199–1210 (2013)
3. Callahan, S., Freire, J., Freire, J., Santos, E., Scheidegger, C., Silva, C., Vo, H.: Managing the evolution of dataflows with vistrails. In: Proceedings of the 22nd International Conference on Data Engineering Workshops, p. 71 (2006)
4. Doedt, M., Steffen, B.: An Evaluation of Service Integration Approaches of Business Process Management Systems. In: Proc. of the 35th Annual IEEE Software Engineering Workshop, SEW 2012 (2012)
5. Ebert, B.E., Lamprecht, A.L., Steffen, B., Blank, L.M.: Flux-P: Automating Metabolic Flux Analysis. Metabolites 2(4), 872–890 (2012), `http://www.mdpi.com/2218-1989/2/4/872`
6. Hammitzsch, M.: The SciForge Project Team: sciforge: Publication and citation of scientific software with persistent identifiers (April 2014), `http://www.sciforge-project.org`
7. Hinkel, J., Vuuren, D.P., Nicholls, R.J., Klein, R.J.T.: The effects of adaptation and mitigation on coastal flood impacts during the 21st century. An application of the DIVA and IMAGE models. Climatic Change 117(4), 783–794 (2012)
8. IIASA/FAO: Global agro-ecological zones (gaezv3.0) (2012), `http://www.gaez.iiasa.ac.at/` (last accessed March 7, 2014)
9. IPCC: Summary for Policymakers. In: Parry, M., Canziani, O., Palutikof, J., van der Linden, P., Hanson, C. (eds.) Climate Change 2007: Impacts, Adaptation and Vulnerability. Contribution of Working Group II to the Fourth Assessment Report of the Intergovernmental Panel on Climate Change, pp. 7–22. Cambridge University Press, Cambridge (2007), `http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Contribution+of+Working+Group+I+to+the+Fourth+Assessment+Report+of+the+Intergovernmental+Panel+on+Climate+Change#6`
10. Jarvis, A., Reuter, H., Nelson, A., Guevara, E.: Hole-filled srtm for the globe version 4 (2008), `http://srtm.csi.cgiar.org/`

11. Kriewald, S.: srtmtools: SRTM tools (2013), r package version 2013-00.0.1
12. Kubczak, C., Jörges, S., Margaria, T., Steffen, B.: eXtreme Model-Driven Design with jABC. In: CTIT Proc. of the Tools and Consultancy Track of the Fifth European Conference on Model-Driven Architecture Foundations and Applications (ECMDA-FA), vol. WP09-12, pp. 78–99 (2009)
13. Lamprecht, A.-L.: User-Level Workflow Design. LNCS, vol. 8311. Springer, Heidelberg (2013)
14. 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), http://www.ncbi.nlm.nih.gov/pubmed/18460174
15. Lamprecht, A.L., Naujokat, S., Margaria, T., Steffen, B.: Synthesis-Based Loose Programming. In: Proc. of the 7th Int. Conf. on the Quality of Information and Communications Technology (QUATIC 2010), Porto, Portugal, pp. 262–267 (September 2010)
16. Lissner, T.K., Reusser, D.E., Schewe, J., Lakes, T.: Linking human well-being and livelihoods with climate change impacts: contextualizing uncertainty in projections of water availability. HESSD (in press, 2014)
17. 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 Systems. Concurrency and Computation: Practice & Experience 18(10), 1039–1065 (2006), http://dx.doi.org/10.1002/cpe.v18:10
18. Ludäscher, B., Altintas, I., Bowers, S., Cummings, J., Critchlow, T., Deelman, E., Roure, D.D., Freire, J., Goble, C., Jones, M., et al.: Scientific process automation and workflow management. In: Scientific Data Management: Challenges, Existing Technology, and Deployment. Computational Science Series, pp. 476–508 (2009)
19. Margaria, T.: Service is in the Eyes of the Beholder. IEEE Computer (November 2007)
20. Margaria, T., Boßelmann, S., Doedt, M., Floyd, B.D., Steffen, B.: Customer-Oriented Business Process Management: Visions and Obstacles. In: Hinchey, M., Coyle, L. (eds.) Conquering Complexity, pp. 407–429. Springer, London (2012), http://books.google.de/books?hl=de&lr=&id=98KSFSfROOEC
21. Margaria, T., Kubczak, C., Njoku, M., Steffen, B.: Model-based Design of Distributed Collaborative Bioinformatics Processes in the jABC. In: Proceedings of the 11th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 2006), pp. 169–176. IEEE Computer Society, Los Alamitos (2006)
22. Margaria, T., Steffen, B.: Agile IT: Thinking in User-Centric Models. In: Margaria, T., Steffen, B. (eds.) ISoLA 2008. CCIS, vol. 17, pp. 490–502. Springer, Heidelberg (2009)
23. Margaria, T., Steffen, B.: Service-Orientation: Conquering Complexity with XMDD. In: Hinchey, M., Coyle, L. (eds.) Conquering Complexity, pp. 217–236. Springer, London (2012), http://dx.doi.org/10.1007/978-1-4471-2297-5_10
24. Naujokat, S., Lamprecht, A.-L., Steffen, B.: Loose Programming with PROPHETS. In: de Lara, J., Zisman, A. (eds.) FASE 2012. LNCS, vol. 7212, pp. 94–98. Springer, Heidelberg (2012)
25. Pradhan, P., Lüdeke, M., Reusser, D.E., Kropp, J.P.: Food Self-Sufficiency across scales: How local can we go? Environmental Science and Technology (under review, 2014)
26. R Core Team: R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria (last accessed March 7, 2014)

27. Steffen, B., Margaria, T., Nagel, R., Jörges, S., Kubczak, C.: Model-Driven Development with the jABC. In: Bin, E., Ziv, A., Ur, S. (eds.) HVC 2006. LNCS, vol. 4383, pp. 92–108. Springer, Heidelberg (2007), http://dx.doi.org/10.1007/978-3-540-70889-6_7
28. Taylor, I., Shields, M., Wang, I., Harrison, A.: The Triana Workflow Environment: Architecture and Applications. In: Workflows for e-Science, ch. 20, pp. 320–339. Springer, New York (2007)
29. Tóth, G., Kozlowski, B., Prieler, S., Wiberg, D.: GAEZ Data Portal - Users's Guide. Tech. rep., IIASA, FAO, Rome, Italy (2012)
30. Vitolo, C., Buytaert, W., El-khatib, Y., Reusser, D.: Big Data for environmental modelling: A review of web technologies. Environmental Modeling & Software (under review, 2014)
31. Warszawski, L., Frieler, K., Huber, V., Piontek, F., Serdeczny, O., Schewe, J.: The inter-sectoral impact model intercomparison project (isi-mip): Project framework. Proceedings of the National Academy of Sciences 111(9), 3228–3232 (2014), http://www.pnas.org/content/111/9/3228.abstract
32. Wilson, G., Aruliah, D., Brown, C.T., Hong, N.P.C., Davis, M., Guy, R.T., Haddock, S.H., Huff, K.D., Mitchell, I.M., Plumbley, M.D., et al.: Best practices for scientific computing. PLoS Biology 12(1), e1001745 (2014)
33. Wrobel, M., Bisaro, A., Reusser, D., Kropp, J.P.: Novel approaches for web-based access to climate change adaptation information – mediation adaptation platform and ci:grasp-2. In: Hřebíček, J., Schimak, G., Kubásek, M., Rizzoli, A.E. (eds.) ISESS 2013. IFIP AICT, vol. 413, pp. 489–499. Springer, Heidelberg (2013), http://dx.doi.org/10.1007/978-3-642-41151-9_45
34. Wrobel, M., Reusser, D.: Towards an Interactive Visual Understanding of Climate Change Findings on the Net: Promises and Challenges. In: Schneider, B., Nocke, T. (eds.) Image Politics of Climate Change, pp. 187–210. Transcript, London (2014), http://www.transcript-verlag.de/978-3-8376-2610-0/image-politics-of-climate-change

A Visual Programming Approach to Beat-Driven Humanoid Robot Dancing

Vid Podpečan[1,2]

[1] Jožef Stefan Institute, Ljubljana, Slovenia
[2] Faculty of Mathematics and Physics, University of Ljubljana, Ljubljana, Slovenia
vid.podpecan@ijs.si

Abstract. The paper presents a workflow-based approach to the classic task of teaching a humanoid robot to dance to a given song with respect to the detected beat. Our goal is to develop workflow components which enable the construction of complex dance choreographies using visual programming only. This eliminates tedious programming of dance moves and their synchronisation to the music thus enabling the robot animator to design the choreography at a higher conceptual level. The presented work is based on the Choregraphe visual programming environment for the NAO robot platform and the Aubio open source tool for the extraction of annotations from audio signals.

Keywords: robot, NAO, dance, workflow, beat.

1 Introduction

Artificial intelligence and mimicking human behaviour and movement are two of the classical goals in the development of humanoid robots. The research in robot artificial intelligence is typically focused on cognitive tasks such as object recognition and interaction with the world, speech recognition and reproduction, learning from sensory data, planning etc. Mimicking of human movement and behaviour includes developing mechanical equivalents of parts of human body (e.g., hands and joints), programming of basic human physical abilities such as fluid body motion, running, jumping, grabbing objects and higher level behaviour such as facial expressions, gestures, postures, and coordinated motion which is typically related to external stimuli, such as sports activities or dance, for example.

The development of self-operating mechanical agents can be traced back to Ancient China, Ancient Greece and Egypt. Knowledge of mechanics, hydraulics and pneumatics [12] was used by ancient engineers to develop machines which were able to perform some relatively simple functions automatically. Between the 17th and 19th centuries, several complex animal and human mechanical automata were built, e.g., mechanical birds, puppets, etc. Although Leonardo da Vinci made plans for a humanoid robot as early as 15th century, the first "true" humanoid robots with a limited set of human-like abilities were built only in the beginning of 20th century. The development of electronic circuits and microprocessors and advances in computer science and mechanics provided the necessary boost for this research field.

Several humanoid robot platforms were developed in the last two decades. They provide sufficient hardware (sensors and mechanical parts) to enable programming of human-like activities. State-of-the-art showcase robots such as ASIMO [13] have achieved a widespread fame for performing complex tasks in front of international audiences. On the other hand, small, affordable robot platforms such as the NAO family were developed to enable research in humanoid robots with the aim to create an intelligent robot companion. Since the introduction of the NAO robot family, a vast amount of research on different topics featuring the NAO robot has been published. In this work, we are interested in programming dancing skills using the NAO platform and open-source software.

Dance is a form of movement involving whole body which is typically - but not always - performed to external stimuli such as rhythm or music. It may include emotional expression and social interaction which are inseparably linked to various types of intelligence. Obviously, for a humanoid robot two separate problems need to be solved: coordinated and aesthetically pleasing movement and expression of individuality and intelligence. While the first problem can be approached using existing methods in robotics, the second one is a computational creativity problem.

In this work we focus on simplifying the task of humanoid robot dance programming by using workflows and audio signal analysis and event extraction. Our approach eliminates programming by providing workflow components which enable visual design of the choreography which is driven by the extracted audio events (onsets). The robot animator can thus focus on perfecting the moves and their logical alignment to a given song.

The NAO platform offers the following options for programming the robot. First, the Choregraphe workflow environment can be used to compose the robot behaviour or action (custom workflow components can also be programmed directly by writing code snippets in the Python language). Second, several programming languages can be used to program the robot directly, e.g., Python, C++, .Net languages, Urbi, etc. As the second solution requires extensive knowledge of robotics and programming, our approach extends the Choregraphe repository with new components which can be used to design a dance choreography using visual programming only. However, an advanced user can modify the components according to her needs or develop new ones.

The rest of the paper is structures as follows. In Section 2 we give an overview of the related work on robot dance programming. Section 3 introduces the Choregraphe workflow environment, describes our approach and its components in details and gives an overview of the recognised limitations. In Section 4 we demonstrate our approach on a simple use case. Section 5 concludes the paper and discusses directions for further work.

2 Related Work

Coordinated and creative robot behaviour has been studied intensively in the last few decades. Due to the vast amount of published research work we limit our discussion to the latest and most similar approaches with respect to the employed robot platform, the general concept and the implementation details.

Conceptually most similar to our approach is the work by Oliviera, Gouyon and Reis [11]. Their framework, based on the Lego Mindstorms NXT platform, tries to simulate dancing behaviour by exracting rhythmic information from audio signals and generate the corresponding behaviour. The Marsyas open source software for audio processing is used to perform onset detection while the developed user interface allows for defining and saving the dance choreography into an XML file.

The approach of Tanaka et al. [15,16] was implemented on the QRIO robot (developed by Sony, now discontinued). The goal was to explore dance interaction between QRIO and children in a classroom environment by detecting and mimicking human movements in order to explore the potential use of interactive robots as instructional tools in education.

The work of Xia et al. [18] extends the idea of robot dance programming with automated planning of a sequence of dance movements which is aligned to the detected beat and emotions. The approach was implemented on the NAO robot platform and extends authors' previous work on extracting beats and emotions from music audio signals.

Shinozaki, Iwatani and Nakatsu [14] have presented an attempt to construct a robot dance system consisted of three units: dance unit sequence generation, dance unit database and dance unit concatenation. They have collaborated with a human dancer and recorded his performance. The selected moves and gestures were replicated on the robot and stored into the database. As the neutral posture is required between two dance units, any sequence of moves can be performed by the robot.

The work of Angulo et al. [1] is targeted at the Sony Aibo animal robot platform where the goal is to create a system which interacts with the user and generates a random sequence of movements for the robot thus creating a human-robot interaction system.

Grunberg et al. [7] have developed an autonomous dancing humanoid system which is also based on beat detection in real time. The beat information is sent to the gesture generation and control system which tries to combine the gestures into a smooth robot motion. The approach, which is a continuation of their earlier work [6], can be deployed on the small, relatively inexpensive Robonova platform or the larger and more complex Hubo humanoid robot.

The goal of the study of Nakaoka et al. [10] is to develop a technology for archiving human dance motions and their reconstruction by a humanoid robot. Their system, deployed on the HRP-1S robot, generates feasible robot leg motions which are based on the recorded human motions.

A different approach to robot dance programming was presented by Aucouturier, Ogai and Ikegami [2]. Instead of preprogrammed or animated dance moves a special type of chaotic dynamics is used to generate robot moves in a seemingly autonomous manner. The generated behaviour is complex but deterministic (as a solution of a non-linear dynamical system) and adapted to the given audio sample. The experiments were performed on the MIURO system, a two-wheeled musical player which can be controlled using a computer.

Finally, the highly influential Evolution of Dance video by Judson Laipply served as an inspiration for the robot version of the same dance[1] (which also went viral) in which the NAO robot was carefully animated to reproduce the human dance performance with great success.

3 Workflow-Based Approach to Robot Dance Programming

The goal of this work is to enable the composition of dance choreographies in the NAO robot workflow environment (Choregraphe, the proprietary NAO visual programming software developed by Aldebaran Robotics). In this section an overview of the approach is given first which is followed by a description of the visual programming environment for NAO. Workflow constituents, which make possible to construct dance choreography workflows are described next. The section concludes with an overview of recognised limitations.

3.1 An Overview of the Approach

Our idea of simplified programming of a dancing robot is based on the concept of a central workflow component, which extracts events from a given audio signal and provides various types of impulses, one time and periodical, which can be used to start and/or stop selected robot behaviours and actions. Consequently, the dance performance can be scripted using appropriate workflow components which perform certain robot actions in a given time frame or beat frame.

The quality of the final robot dance performance is based on: (a) the accuracy of the extracted audio events, (b) the quality and size of the library of animated robot actions, (c) the quality of arrangement of robot actions, and (d) fine-tuning of the animated robot actions for the particular musical piece.

Developing robot dance choreography in a workflow environment introduces few important advantages (which are not specific to robot programming but are valid for any workflow-based solution). First, once the basic building blocks are implemented, no expert knowledge is needed to compose a solution or new use cases (in our case, few basic components are already provided by Choregraphe while the majority was developed from scratch). Second, different values of parameters and choreography arrangements can easily be tested. Third, easy sharing of the complete solutions is one of the key features of workflows. Finally, debugging and explanations of the implemented steps and procedures are

[1] `http://www.youtube.com/watch?v=2laujomh0JY`

greatly simplified. However, the ease of use and simplicity also introduce certain drawbacks. The most notable limitations of our approach are discussed in Section 3.4.

To our best knowledge the presented work is the first attempt to create a complete robot dance choreography using an executable workflow[2]. Most typically, the performance is either implemented in a single program or composed of few interconnected modules. For example, Oliveira et. al [11] connect the music analysis module built with Marsyas [17] with a simple, manually developed graphical user interface. Ellenberg et. al [6] have also developed a simple graphical user interface which serves as a glue for the implemented audio processing algorithms and robot control routines. Both graphical user interfaces are intuitive and seem relatively flexible, however, they cannot compare to a fully featured visual programming environment.

3.2 The Choregraphe Visual Programming Environment

Choregraphe is a multi-platform application which is a part of the NAO robot platform that enables creating complex robot behaviour without writing program code. It allows to develop animations and behaviours and test them on the real or the simulated robot. The application is based on the visual programming paradigm. It provides a wide range of visual programming components, e.g., flow control, basic robot motions, audio and speech, robot sensing, robot vision etc. New components of three different types can be added: *script*, *timeline* and *flow diagram*.

The first type allows for advanced visual programming where the behaviour of the component and the manipulation of its inputs and outputs is implemented in the Python programming language. The second type enables designing complex robot motion using the timeline editor, recording mode and animation mode. Finally, the flow diagram allows for nested diagrams which is a very useful feature when designing large and complex procedures.

The granularity of visual programming components in Choregraphe is somewhat finer than in a typical scientific workflow system. Data mining and knowledge discovery workflows in various scientific domains are usually constructed from several high-level data manipulation and visualisation steps, such as data parsing, missing value handling, noise and outlier detection and removal, discretisation, dimensionality reduction and visualisation, etc. because the majority of data mining workflow software is designed to be data-driven. For example, workflow environments such as KNIME [3], RapidMiner [9] and Taverna [8] support looping and conditional execution to some extent but in most cases their presence in a data mining workflow indicates inappropriate design of the solution or a problem which is structurally too complex for a single workflow.

On the other hand, workflows constructed in Choregraphe are event-based. Event signals may carry additional data but in most cases only trigger some

[2] It is also true that only few visual programming environments for robots exist. Besides Choregraphe, Microsoft Robotics Developer Studio is an example of a fully featured programming environment for building robotics applications.

specific action or excite the connected input. The Choregraphe visual programming environment was designed in a way which simplifies robot programming and our approach is based on its features. For a detailed overview of the Choregraphe software we refer the reader to the official documentation[3]. Here we only summarise the most important facts which are relevant for further discussion:

- the communication between workflow components is event based,
- the event signal, which is sent from one component to another, can also carry information,
- several input/output types are available:
  1. bang: a simple event without data
  2. number: a float or an int or an array of numbers
  3. string: a string or an array of strings
  4. dynamic: either a simple event (bang) or an event carrying data (number, string, array of numbers, strings and arrays)
- new workflow components can be programmed by providing code snippets in Python,
- workflow components have full access to the Python language environment and the NAO robot operating system NAOqi,
- meta workflows are supported (workflow components can be grouped and converted into a single workflow component which contains a nested subworkflow).

Because Choregraphe's workflows can grow quite large and complex when implementing an elaborate robot behaviour, nested workflows can be used to organise and group the components into a hierarchical structure.

3.3 The Developed Workflow Components

As our goal is to enable robot dancing which is aligned to the detected beat, the most important workflow component is the *beat detector* which performs the analysis of a given audio signal and detects onsets, i.e., high energy peaks. For the audio analysis task we have employed the Aubio open source tool [4,5] which is designed for the extraction of annotations from audio signals. More specifically, the *aubioonset* component which extracts musical onset times is used. Aubio has no mandatory software dependencies which makes it a perfect choice for the optimised Linux distribution which runs on the NAO robot. We have compiled the library using the OpenNAO virtual machine, made available by Aldebaran Robotics[4].

We have integrated *aubioonset* using two workflow components as follows. The first component receives an audio file in the PCM wave format and runs *aubioonset* from Python. It supports the same main extraction options as the

[3] `https://community.aldebaran-robotics.com/doc/1-14/software/choregraphe/index.html`

[4] Alternatively, any Linux distribution with GNU gcc compiler can be used to cross-compile for NAO.

command line utility: (a) onset detection method, (b) onset threshold, (c) silence threshold, (d) buffer size, and (e) hop size. The first three parameters are the most important as they enable fine tuning for different audio signals. The component returns a Pyhon array of detected time points marking musical onset times.

The second component which is actually the "heart" of the workflow, receives the computed array of onset time points and generates periodic and/or one-time events. For periodic beat-based events the component provide several downsampled outputs (e.g., 1x, 2x, 4x, 8x, etc.) but new, custom outputs can also be added. Events for the beginning and end of a specified time period are also available.

Another important workflow component is *selector* which activates one of its outputs according to the specified user preferences. For example, when triggered, the component can randomly choose one of the outputs and activates it by sending the *bang* signal. Several selectors can be used in a workflow in order to group similar behaviours (a selector hierarchy can be also easily constructed in order to organise the workflow on the logical level). While randomisation is currently the only available selection method, we are planning to develop a more intelligent approach which will be based on the emotions extracted from the audio sample.

The *allow n times* component is a generalisation of the Choregraphe's *only once* component. It allows the signal to pass n-times. Its typical usage in the workflow is to allow only n repetitions of a selected robot action or actions.

Few general dance gestures and motions were also developed using Choregraphe's recording capabilities. The recording mode enables the user to record the positions of all movable robot parts and stores them for editing and replay but also offers export in the form of raw numerical and time data. Using the recording mode, we have produced several workflow components of the type "timeline" which are performed by the robot when selected by the *selector* component during the performance. 9 gestures are available (see Figure 1 for an example):

– both hands move above the head and back
– swimming motion with both hands
– left (right) palm moves across the face
– left (right) hand does a full circle at the chest level
– short knee motion
– "yes" and "no" head motions

Finally, an extensive repertoire of Choregraphe's built-in components is available and can be used to: (a) program certain robot actions such as such as light signals with built-in LEDs (eyes, ears and chest button) and audio input/output, (b) control the flow of execution using if, for, switch, stop and other control structures, (c) make use of sensor data and (d) handle communication (IR and network).

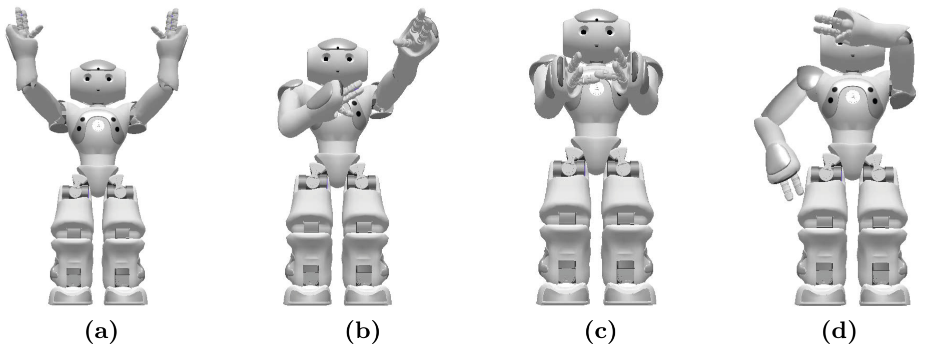


Fig. 1. Few samples of the NAO robot model performing dance gestures. When the dance choreography workflow is running, gestures are performed according to the extracted beat and selected at random using the *selector* component.

3.4 Limitations

Because the proposed workflow-based approach is intuitive, easy to use and based on existing software, this also introduces certain limitations.

The *aubioonset* tool from the aubio library only provides accurate onset detection (local onset peaks) but does not try to compute the global tempo. This can lead to undesired robot behaviour, e.g., rhythmical motions are interrupted during longer silent periods.

Although a simple beat-driven dance choreography workflow can be constructed in minutes, the resulting dance will indeed seem "robotic" as certain behaviour will be repeated continuously (unless some clever randomisation method is employed). On the other hand, long and complex choreography workflows may introduce the problem of synchronisation. Small, insignificant delays which are unavoidable when executing workflow components may add up to a significant delay which can cause the robot to become desynchronised with the rhythm. This is especially important for very fast beats such as those found in uptempo electronic dance music.

Currently, the adaptation of the speed of robot moves for a particular musical genre has to be done manually as the detected onsets are only used to initiate robot actions, not to set their execution time frame. However, although the animations of robot moves are measured in time frames, they can be easily compressed or expanded given the number of beats the actions should take and the global value of beat per minute. The dynamic adaptation of the duration of robot actions is left for future work.

Finally, our approach is targeted solely to the NAO robot platform. While the components can be easily reprogrammed for different robot hardware, the most attractive part, namely the dance workflow composition, requires a capable workflow execution environment which is able to communicate with the robot platform.

4 Experiments

We have performed experiments with the developed approach on a few audio samples. A classic 80's disco song "Hands Up (Give Me Your Heart)" by Ottawan was selected as a test case as it features a regular beat at moderate speed and the appropriate disco dance gestures are both recognisable and easy to animate.

Figure 1 shows a 3D model of the NAO robot performing few selected dance gestures. In this simple experiment, only the developed basic gestures (see Section 3.3) and built-in light effects were used. For a more elaborate dance choreography, new gestures have to be recorded or programmed and stored into separate workflow components.

The workflow which implements a simple dance to the song mentioned above is shown in Figure 2. It can be easily extended by providing new gestures and moves and adding *selector* and *allow n times* components to control the performance. The workflow works as follows. From a high level perspective, it is composed of three parts: the preparatory steps, the dance performance and the conclusion of the performance.

The preparatory phase begins with instructing the robot to go to the predefined "StandInit" posture from which he is able to perform any action or move. The next component moves the hands to a low position which is the starting position for dance moves. The last two preparatory steps load the audio file on the robot and run the *aubioonset* tool to extract the beat. The resulting array of float values indicating the beat onset times is then send to the core component of the workflow, the beat emitter.

The main part of the workflow (the dance performance) consists of several timeline components containing animated robot dance moves, few selector components and the beat emitter. The beat emitter contains a loop which periodically emits signals on its outputs. Several outputs are already available but Choregraphe software also allows for adding and modifying new inputs/outputs on the fly. In the current setup, 6 user-configurable outputs are available. Each of them can emit the beat signal with the period from 1 up to 1024 beats. Outputs with the longer period are used to trigger robot moves with long duration (raising both hands, swimming motion, etc.) while the ones with the short period control fast events such as blinking, nodding and contraction of knees. 4 selector components are used to group dance gestures into logical groups in order to organise the structure of the workflow. Note that for the adaptation of the workflow to another musical piece there are no mandatory changes (from the technical perspective). However, in order to achieve an aesthetically pleasant performance, specific moves and gestures have to be developed which are suitable for the chosen musical piece. Also, the current setup is rather simple and does not take into account the structure of the musical piece: introduction, verse, chorus, etc.

The last part of the workflow concludes the performance as follows. First, the robot is put again into the stable "StandInit" posture. The animated forehead wiping is performed next which is followed by bowing to the audience and blowing a kiss. Finally, the robot is put into the sitting position and the motors of his joints are turned off.

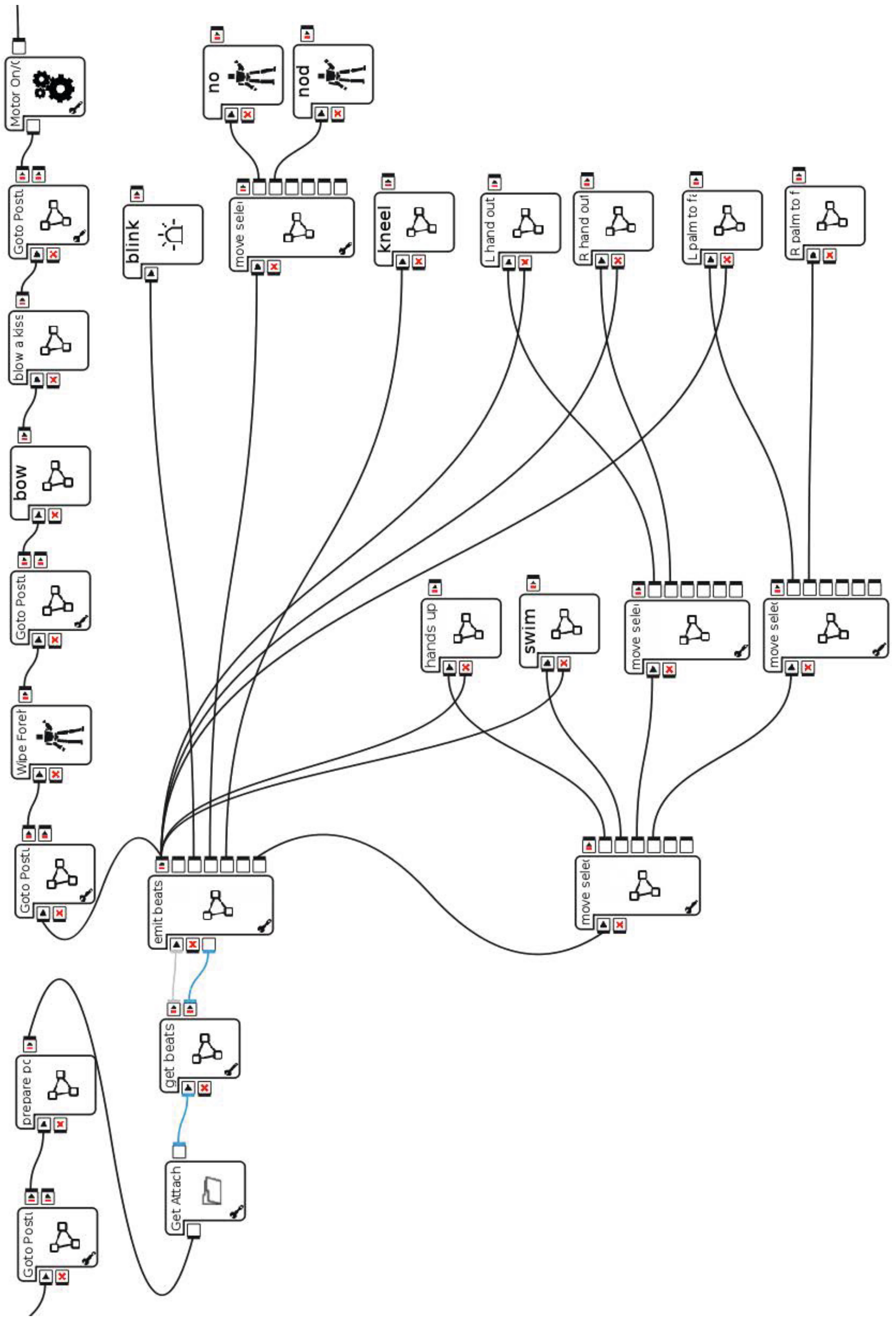


Fig. 2. A workflow in the Choregraphe virtual programming environment implementing a simple dance using the developed components and few animated robot actions

The majority of workflow components used in this experiment were developed from scratch. With the exception of few general supporting and basic robot controlling component such as "Goto posture", "Motor on/off" and "Load file", the visual programming elements were either implemented (script components) or animated (timeline components). Examples of the former are e.g., beat detector, beat emitter and move selector while the examples of the latter are dance gestures such as "both hands up", "left palm to face", "right hand out", "nod", etc.

5 Conclusions and Further Work

In this paper we have presented an approach which enables implementing beat-synchronised dance choreographies for the NAO robot platform using visual programming in the Choregraphe software environment. The developed workflow components allow the robot animator to compose a dance choreography in very short time without programming and/or expert knowledge of the robot software interfaces.

The approach is simple and intuitive and can give satisfactory results with little effort as only general knowledge about visual programming using the Choregraphe environment is required in order to construct a simple dance choreography. However, there are several limitations which have to be taken into account (see Section 3.4 for an overview). As our main future goal, we plan to improve onset detection to produce the accurate global tempo instead of local peaks. This will improve the dance performance enormously as the robot events will be synchronised to the beat as it is perceived by humans.

Furthermore, in order to improve the individuality of a performance we are investigating options to detect emotions from audio in order to intelligently select an appropriate gesture. Employing natural language processing techniques to extract additional information from lyrics is also an interesting option.

An independent evaluation of the developed dance performances is also required in order to be able to compare to the existing related work in formal terms. This also implies the development of several new dance choreographies.

Finally, we will create an extensive library of NAO robot gestures and moves for different music styles and develop automated music classification by genre which will be used to select the appropriate family of robot moves from the library.

Acknowledgements. This work was supported by FP7 projects WHIM (The What-if Machine) under the Grant Agreement No. 611560 and MUSE (Machine Understanding for interactive StorytElling) under the Grant Agreement No. 296703.

References

1. Angulo, C., Comas, J., Pardo, D.: Aibo jukeBox – A robot dance interactive experience. In: Cabestany, J., Rojas, I., Joya, G. (eds.) IWANN 2011, Part II. LNCS, vol. 6692, pp. 605–612. Springer, Heidelberg (2011)
2. Aucouturier, J.-J., Ogai, Y., Ikegami, T.: Making a robot dance to music using chaotic itinerancy in a network of fitzhugh-nagumo neurons. In: Ishikawa, M., Doya, K., Miyamoto, H., Yamakawa, T. (eds.) ICONIP 2007, Part II. LNCS, vol. 4985, pp. 647–656. Springer, Heidelberg (2008)
3. 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 (2007)
4. Brossier, P.M.: Automatic Annotation of Musical Audio for Interactive Applications. Ph.D. thesis, Centre for Digital Music, Queen Mary, University of London (August 2006)
5. Brossier, P.M.: Aubio, a library for audio labelling (2014), `http://aubio.org/`
6. Ellenberg, R., Grunberg, D., Kim, Y., Oh, P.: Exploring creativity through humanoids and dance. In: Proceedings of the 5th International Conference on Ubiquitous Robots and Ambient Intelligence, URAI 2008 (November 2008)
7. Grunberg, D.K., Ellenberg, R., Kim, I.H., Oh, J.H., Oh, P.Y., Kim, Y.E.: Development of an autonomous dancing robot. International Journal of Hybrid Information Technology 3(2) (2010)
8. Hull, D., Wolstencroft, K., Stevens, R., Goble, C.A., Pocock, M.R., Li, P., Oinn, T.: Taverna: A tool for building and running workflows of services. Nucleic Acids Research 34(Web-Server-Issue), 729–732 (2006)
9. Mierswa, I., Wurst, M., Klinkenberg, R., Scholz, M., Euler, T.: Yale: Rapid prototyping for complex data mining tasks. In: Ungar, L., Craven, M., Gunopulos, D., Eliassi-Rad, T. (eds.) KDD 2006: Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 935–940. ACM, New York (2006)
10. Nakaoka, S., Nakazawa, A., Yokoi, K., Ikeuchi, K.: Leg motion primitives for a dancing humanoid robot. In: Proceedings of the IEEE International Conference on Robotics and Automation, ICRA 2004, vol. 1, pp. 610–615 (April 2004)
11. Oliveira, J., Gouyon, F., Reis, L.P.: Towards an interactive framework for robot dancing applications. In: International Conference on Digital Arts, Porto, Portugal (2008)
12. Rosheim, M.: Robot Evolution: The Development of Anthrobotics. Wiley interscience publication, Wiley (1994)
13. Sakagami, Y., Watanabe, R., Aoyama, C., Matsunaga, S., Higaki, N., Fujimura, K.: The intelligent ASIMO: System overview and integration. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, vol. 3, pp. 2478–2483 (2002)
14. Shinozaki, K., Iwatani, A., Nakatsu, R.: Concept and construction of a dance robot system. In: Proceedings of the 2nd International Conference on Digital Interactive Media in Entertainment and Arts, DIMEA 2007, pp. 161–164. ACM, New York (2007)

15. Tanaka, F., Fortenberry, B., Aisaka, K., Movellan, J.R.: Plans for developing real-time dance interaction between QRIO and toddlers in a classroom environment. In: Proceedings of the 4th International Conference on Development and Learning, pp. 142–147 (July 2005)
16. Tanaka, F., Suzuki, H.: Dance interaction with QRIO: A case study for non-boring interaction by using an entrainment ensemble model. In: 13th IEEE International Workshop on Robot and Human Interactive Communication, ROMAN 2004, pp. 419–424 (September 2004)
17. Tzanetakis, G., Cook, P.: Marsyas: A framework for audio analysis. Organized Sound 4, 2000 (2000)
18. Xia, G., Tay, J., Dannenberg, R., Veloso, M.: Autonomous robot dancing driven by beats and emotions of music. In: Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2012, vol. 1, pp. 205–212. International Foundation for Autonomous Agents and Multiagent Systems, Richland (2012)

jABCstats: An Extensible Process Library for the Empirical Analysis of jABC Workflows

Alexander Wickert and Anna-Lena Lamprecht

Chair for Service and Software Engineering, University of Potsdam,
August-Bebel-Str. 89, 14482 Potsdam, Germany
`{awickert,lamprecht}@cs.uni-potsdam.de`
`http://www.cs.uni-potsdam.de/sse`

Abstract. The jABC is a multi-purpose modeling framework that has been used for model-driven development of workflows and processes in different application domains. In this paper we present jABCstats, an extensible process library for analyzing jABC workflows empirically. We also discuss first results of its application to scientific workflows modeled with the jABC, which give insights into typical workflow sizes and into the kinds of services and the workflow patterns commonly used.

Keywords: jABCstats, scientific workflows, model-driven development, jABC, service usage, workflow motifs, workflow patterns.

1 Introduction

The notion of scientific workflows has been used as a concept for bridging the (semantic) gap between IT and scientific applications for more than a decade now. Numerous systems with different characteristics have been developed for supporting the design, management and execution of scientific workflows, and new ones continue to be developed. At the same time, a tremendous amount of scientific workflows has been designed with these systems, and provides a rich source for the systematic analysis of scientific workflows (cf., e.g., [4,11,21]). A better understanding of their characteristics can in fact give direction to the improvement of scientific workflow management systems in the future.

In our work with scientific processes (cf., e.g., [3,7,8,9,12,13]), we use the jABC framework [20] as workflow management system and the jETI technology [14] for remote service integration. The jABC is a multi-purpose and domain-independent modeling framework that is primarily used for user-centric development of process and workflow applications according to the XMDD (eXtreme Model-Driven Design) paradigm [15,16]. Users simply create hierarchical models called Service Logic Graphs (or SLGs) from collections of reusable workflow building blocks, called Service-Independent Building Blocks (or SIBs) via an intuitive graphical user interface. The SIBs encapsulate accessible units of functionality, which can be calls to library functions or remote services, as well as auxiliary functionality such as basic data manipulations or condition evaluations. The SLGs are flow-graph structures that graphically represent the flow of control, that is, the order in which the SIBs are executed. Figure 1 gives an impression of how the individual SIBs form an SLG on the modeling canvas and how workflow execution is animated by the built-in interpreter.

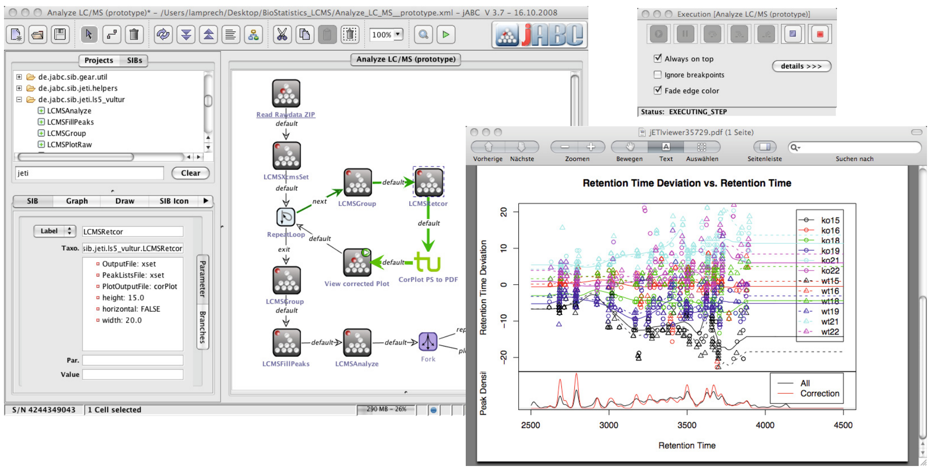


Fig. 1. Graphical user interface of the jABC framework

In fact, the SLGs provide a comprehensive and powerful modeling formalism, so that parts of the jABC have themselves been developed as jABC workflows (e.g., [6,17]). This paper describes another self-application called jABCstats. It is an extensible library of processes (realized as jABC workflows) that provides functionality for empirical analysis of jABC workflows. The statistics that can currently be generated by jABCstats provide detailed counts of all SIBs (both individually and package-wise) and of selected workflow patterns that are used by the examined workflows.

We used the library to analyze a first selection of scientific workflows modeled with the jABC. The workflows were created by computer science, bioinformatics and geoinformatics students in the scope of three courses on workflow modeling that we taught in the last terms. Note that the workflows were not prescribed, the students were themselves responsible for defining the (scientific) process applications for their projects. Hence, this collection appears to be an adequate sample for studying general characteristics of scientific workflows. The analysis results that we obtained from this first sample give insights into typical workflow sizes and into the kinds of services and the workflow patterns commonly used.

The paper is structured as follows: Section 2 describes the extensible process library and how its workflows perform the analyses. Section 3 presents some first results from the analysis of scientific workflows modeled in the jABC, discusses our findings and compares them to related work. Section 4 closes the paper with a summary and ideas for future work.

2 Extensible Process Library for Generating Statistics

This section describes the main workflows of the process library that creates statistics about jABC workflows. Note that the process library was completely modeled using the standard SIB library that is shipped with every jABC distribution, that is, there was neither the need to implement any new SIBs, nor the need to call scripts or (web) services. To give an overview of the processes contained in the library and their relationships, Figure 2 visualizes the hierarchy of all process models. It easily can be seen that the maximum model depth is 4, but one can also see that the four main processes (re-)use other models. For instance, `_Main_GetSingleModelStatistic` and `_Main_GetSeveralModelStatistics` both contain the model `GetModelStatistics` as a submodel.

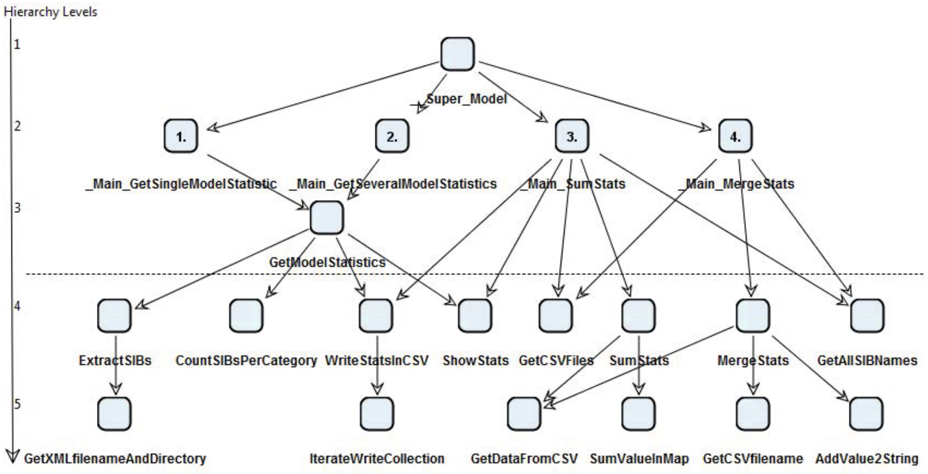


Fig. 2. The hierarchy of the models of the process library

Due to the limited space in this paper, we will in the following only describe the three top levels of this model hierarchy. Currently, four different (extensible) core processes are available, as summarized in Table 1 (Note that subdirectories will always be ignored and only *.csv files created with workflow no. 1 or 2 are supported as valid inputs for workflow no. 3 and 4.):

1. a process for generating the statistic for only one single jABC model file,
2. a process for generating the statistics for several jABC model files,
3. a process for summarizing several *.csv statistics files, and
4. a process for merging several *.csv statistics files.

The following descriptions of jABC workflows always go from left to right and from top to bottom. The super model depicted in Figure 3 is the most general model, prepared to call all four core workflows. It starts with the SIB

Table 1. Overview of the core processes and their inputs and outputs

Process	Inputs	Outputs
1. Generating single statistic	One single jABC model *.xml file.	Corresponding *.csv file with statistic.
2. Generating several statistics	Directory with several jABC model *.xml files.	Directory with all corresponding *.csv files with statistics.
3. Summarizing statistics	Directory with several *.csv files.	One single *.csv file containing total values summarized for all SIBs and their namespace hierarchies in one row.
4. Merging statistics	Directory with several *.csv files.	One single *.csv file, where every row contains the numbers of one input *.csv file (extended over all SIBs and their namespace hierarchies of all *.csv files).

`PutBoolean - interactive` (with the underlined caption). The first three SIBs define three Boolean values:

1. **interactive** – If true, messages will be displayed during workflow execution.
2. **createFile** – If true, an output file will be created on the local file system.
3. **showStats** – If true, the generated statistics will be shown.

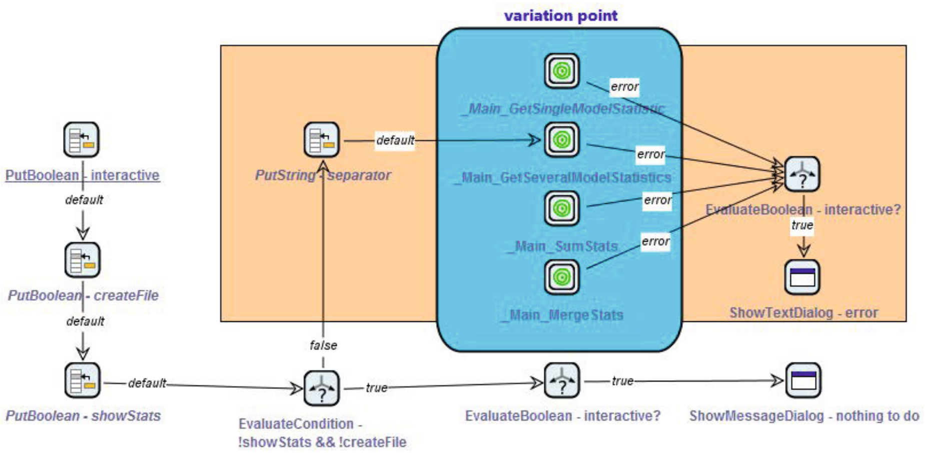


Fig. 3. General model with variation point for the four core functionalities (__Super_Model)

Next, the SIB `EvaluateCondition` tests the condition whether the statistics should not be shown and the file should not be created. This catches just the case when the user sets the parameters in this senseless way. If this is the case (`true` branch), the workflow will terminate and - if interactive is true - will show a message that nothing is to do. Otherwise (`false` branch), the workflow goes on with `PutString - separator` (in the rectangle) that sets the separator of the *.csv file (default is semicolon `;`). The following rounded rectangle marks a variation point (cf., e.g., [19]) in the model. At this point, different submodels (representing the four core processes described above) can be chosen. Therefore one simply has to connect the outgoing branch of `PutString - separator` to the SIB representing the process of choice. In Figure 3 it is currently connected to `_Main_GetSeveralModelStatistics`, thus only this process will be executed at runtime of this model. All other SIBs in the rounded rectangle are alternatives. If any error occurs during the execution of the submodel and interactive is set true, then an error message will be shown to the user.

2.1 A Process for Generating the Statistic for Only One jABC Model

In the following, the process for generating the statistic for only one single jABC model (Figure 4) will be explained in more detail. This model is in fact very simple. There is only one SIB that sets the path to the current jABC *.xml model file and a second SIB that calls the next submodel `GetModelStatistics`. The reason for aggregating the workflow this way is simply that it facilitates the reuse of `GetModelStatistics` that is also used in the next process, described in Section 2.2.

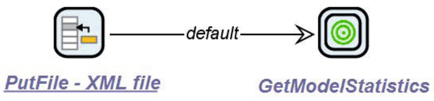


Fig. 4. Main model for generating statistics for only one jABC model (_Main_GetSingleModelStatistic)

Figure 5 takes a closer look at the `GetModelStatistics` process. The first component `ExtractSIBs` extracts all SIBs that are stored in the jABC *.xml file using a regular expression and stores them in a collection. The code snippet below shows how a SIB description in the jABC model *.xml file looks like:

```
<sib>
  <id>a4312a58-6825-4835-ac80-2a62e8f906eb</id>
  <name>MacroSIB</name>
  <label>_Main_GetSingleModelStatistic</label>
  <taxonomy>de.metaframe.jabc.sib.MacroSIB</taxonomy>
  ...
</sib>
```

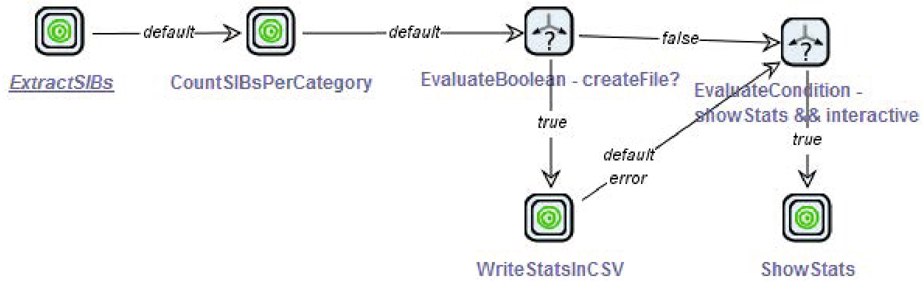


Fig. 5. Actual model for generating statistics for a jABC model (GetModelStatistics)

The full SIB name (namespace and concrete SIB) that is relevant for the analysis is located between the tags `<taxonomy>` and `</taxonomy>`. In the above example, `de.metaframe.jabc.sib.MacroSIB` is extracted and the complete namespace is divided into all of its hierarchy levels.

Then, in `CountSIBsPerCategory`, all SIBs and all of their partial namespaces are counted and put into a sorted map, where the key is the partial namespace/complete SIB name and the value is the number of occurrences. `EvaluateBoolean - createFile?` tests whether the user selected to create a *.csv file from the results of `CountSIBsPerCategory`. If yes, `WriteStatsInCSV` writes the results into a *.csv file, where the first row contains the partial namespaces/complete SIB names and the second row contains the corresponding numbers of occurrences. Table 2 shows some columns of the resulting output *.csv file of `GetModelStatistics`. One can see that the occurrences of SIBs in all namespace levels are counted. The total number of SIBs in the analyzed model is thus 13, and all belong to the `de.*` package. There are 4 `MacroSIBs`, but no other SIBs of package `de.metaframe.*` are used in `__Super_Model`, thus the counts for namespace `de.metaframe.*` and all packages inside it are exactly 4.

Finally, `EvaluateCondition - showStats && interactive` tests if `showStats` and `interactive` are `true`. Only in this case (`true` branch), `ShowStats` shows the resulting statistics in a dialog window to the user.

Table 2. Snippet of statistical output of GetModelStatistics

#allSIBs	de.*	de.metaframe.*	de.metaframe. jabc.*	de.metaframe. jabc.sib.*	de.metaframe. jabc.sib.MacroSIB	...
13	13	4	4	4	4	...

2.2 A Process for Generating Statistics for Several jABC Models

The process depicted in Figure 6 generates statistics for several jABC model *.xml files. As indicated in the previous section, decomposing a model into different submodels allows one to reuse them easily, and so the only necessary work to create the functionality for generating statistics for several models was in fact to model an iteration around `GetModelStatistics`. The output *.csv files are exactly like in Table 2.

At first, `PutFile - XML dir` defines the directory that contains all jABC *.xml files. `ScanDirectory` puts all *.xml files into a collection and `GetSize` gets the size of this collection (= number of *.xml files). The next condition tests, if the number of *.xml files is greater than 0. If not (`false` branch) and interactive is `true`, a dialog will be shown to the user that the chosen directory contains no *.xml files. Otherwise (`true` branch of `EvaluateCondition`), the *.xml files will be iterated and for each file `GetModelStatistics` will be executed (see rectangle). (Note that until this point there is no check whether the .xml files are valid jABC models. If it is not a valid jABC model, then the file will be skipped. Also errors in the submodel will not terminate the whole process.) If `interactive` is `true`, a final message informs the user that the complete process is finished.

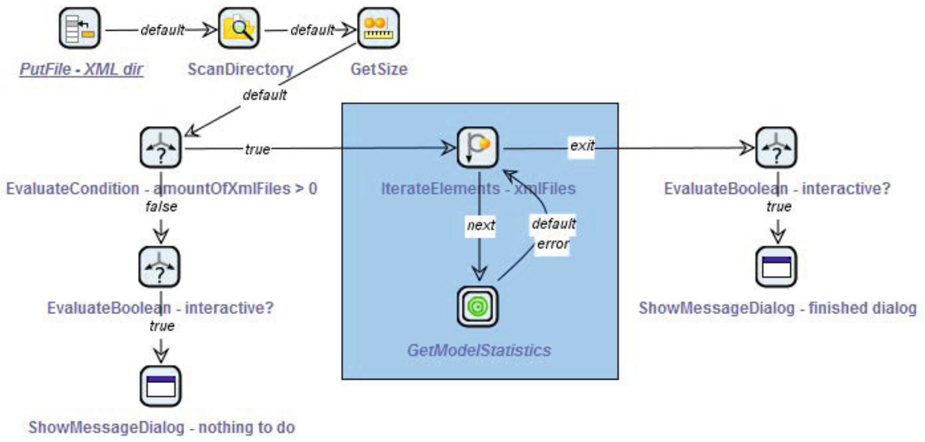

PutFile - XML dir
default
ScanDirectory
default
GetSize
default
EvaluateCondition - amountOfXmlFiles > 0
true
false
EvaluateBoolean - interactive?
true
ShowMessageDialog - nothing to do
IterateElements - xmlFiles
next
default
error
GetModelStatistics
exit
EvaluateBoolean - interactive?
true
ShowMessageDialog - finished dialog

Fig. 6. Main model for generating statistics for several jABC models (_Main_GetSeveralModelStatistics)

2.3 A Process for Summarizing Statistics

As a consequence of the ongoing work with the examined workflows and the work with the created *.csv files, ideas for further functionalities were developed. One idea that already has been realized is described in this subsection and another one in Section 2.4.

The first idea was to have a process that is able to take several *.csv files (as created by the processes described above) as input and summarize the numbers of all columns automatically. The resulting output is a new *.csv file that contains again exactly two rows like in Table 2. With this functionality, one could e.g. summarize the statistics of all individual workflows of one project in a single *.csv file.

The process for summarizing statistics of several *.csv files can be seen in Figure 7. It starts with `GetCSVFiles` that retrieves all *.csv files as a collection from the input directory. Then, `GetAllSIBNames` extracts all SIB and package names from every input *.csv file. Afterwards, `SumStats` summarizes the numbers of all SIBs and all (partial) namespaces to total numbers. The rest of the workflow is like in Figure 5.

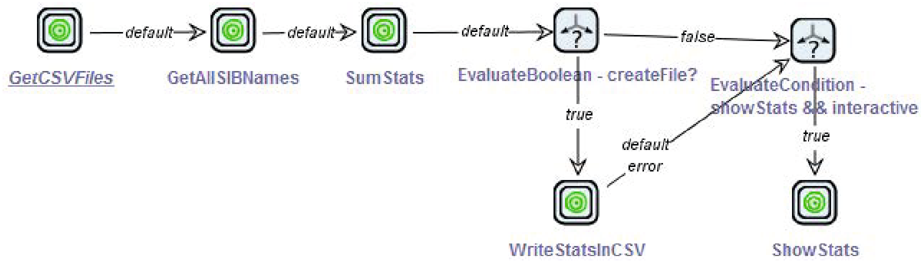


Fig. 7. Main model for summarizing statistics from several *.csv files (_Main_SumStats)

2.4 A Process for Merging Statistics

The last functionality merges statistics of several *.csv files into one *.csv file, with the purpose to give a complete overview of different workflow projects or models. Therefore, every row in the output table contains the numbers of one input *.csv file, and the columns, as before, cover all SIBs and their namespace hierarchies. A snippet from an example output *.csv file generated by this process (applied on the process itself) can be seen in Table 3. The resulting merged *.csv file contains a lot of columns, especially some of them contain many `0`s (like `CheckVariable`) because this SIB was only used in few models. Hence, the output can get very big when many different SIBs have been used in the different projects. Consequently, the execution time of this workflow increases also very fast, thus we will have to investigate in optimization possibilities here as the development and use of the library continues.

Figure 8 depicts the model of this process. It starts (like in Figure 7) with `GetCSVFiles` that retrieves all *.csv files as a collection from the input directory. `GetAllSIBNames` then extracts all SIB names from every input *.csv file. Afterwards, `MergeStats` merges the stats of all input *.csv files into one single *.csv file extended over all SIBs and their namespace hierarchies of all *.csv files. The way of writing the *.csv file (`WriteTextFile`) and showing the results (`ShowTextDialog`) differs from the previous processes. The submodels are replaced by atomic SIBs because (for performance reasons) the string that contains the output content of the *.csv file is already constructed while merging the statistics.

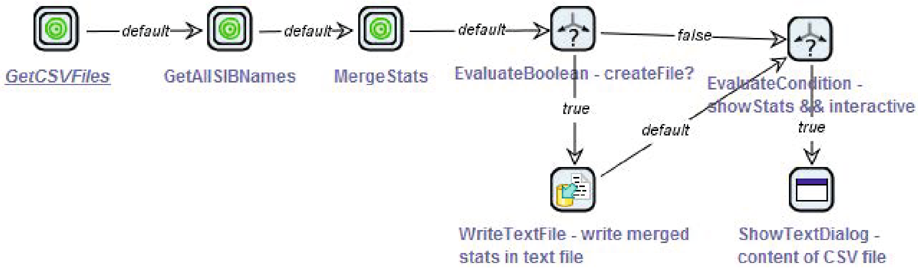

GetCSVFiles
default
GetAllSIBNames
default
MergeStats
default
EvaluateBoolean - createFile?
false
true
EvaluateCondition - showStats && interactive
default
true
WriteTextFile - write merged stats in text file
ShowTextDialog - content of CSV file

Fig. 8. Main model for merging statistics from several *.csv files (_Main_MergeStats)

Table 3. Snippet of statistical output of _Main_MergeStats

name of CSV file	#allSIBs	de.*	de. jabc.*	de. jabc. sib.*	de. jabc. sib. common.*	de. jabc. sib. common. basic.*	de. jabc. sib. common. basic. CheckVariable	de. jabc. sib. common. basic. PutFile	...
_Main_GetSeveral-ModelStatistics	10	10	9	9	9	4	0	1	...
_Main_GetSingle-ModelStatistic	2	2	1	1	1	1	0	1	...
_Main_MergeStats	7	7	4	4	4	2	0	0	...
_Main_SumStats	7	7	2	2	2	2	0	0	...
GetModelStatistics	6	6	2	2	2	2	0	0	...
...	...	...	...	...	...	...	...	...	...

2.5 Extensions of the Current Process Library

Extensions of the current process library are easily possible due to its hierarchical and modular structure. Figure 9 exemplarily shows how two new submodels, `GetMaxModelDepth` and `AnalyzeModelPatterns`, can be integrated into the `GetModelStatistics` workflow. They have not yet been modeled completely, but we are working on them. `GetMaxModelDepth` is going to automatically determine the maximal depth of a jABC workflow (i.e. the maximal number of hierarchy levels) and `AnalyzeModelPatterns` is going to count typical control-flow patterns in the workflow, such as sequences, choices, forks and loops.

3 First Results

As a first study with our new process library for empirical analyses of workflows, we analyzed a set of 54 scientific workflows projects consisting of 241 individual workflow models that were created by computer science, bioinformatics and geoinformatics students in the scope of three Master-level courses on workflow

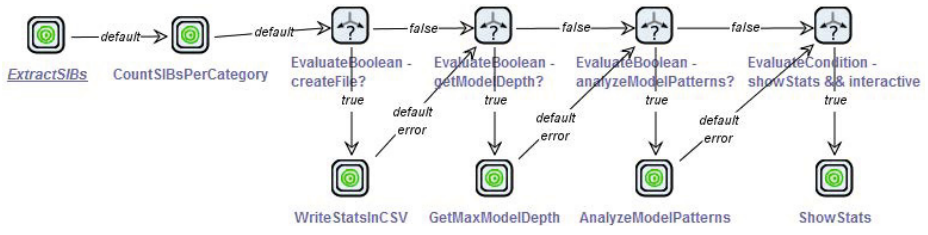

ExtractSIBs
default
CountSIBsPerCategory
default
EvaluateBoolean - createFile?
false
EvaluateBoolean - getModelDepth?
false
EvaluateBoolean - analyzeModelPatterns?
false
EvaluateCondition - showStats && interactive
true
default
error
WriteStatsInCSV
GetMaxModelDepth
AnalyzeModelPatterns
ShowStats

Fig. 9. Variant of GetModelStatistics with two new analysis functionalities

modeling that we taught in the last terms. Note that the workflows were not prescribed at all, but the students (as Master students are already advanced in their respective fields) were themselves responsible for defining the scientific process applications of their projects. As a consequence, the workflow projects deal with a wide range of scientific topics, and accordingly a large variety of tools and services is used within the workflows. As such, this collection appears to be an adequate sample for studying general characteristics of scientific workflows. As detailed in the following, our first findings largely comply with results from previous empirical studies on scientific workflows and substantiate different impressions that we got in our and the students' work with scientific workflows.

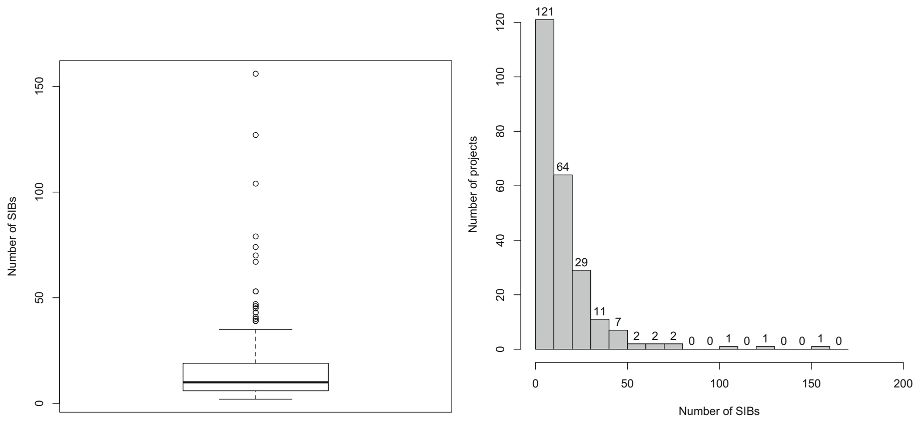


Fig. 10. Size of the examined workflows with respect to the number of used SIBs

In our sample, the size of the workflows (simply measured by the total number of SIBs in the workflow model(s)) ranges from 2 to 156, with a mean of 15.9, a median of 10 and a standard deviation of 18.3, as visualized by the box plot in Figure 10 (left). The histogram in Figure 10 (right) shows in more detail that in fact most of the workflows (121) consist of only up to 10 SIBs, about a quart (64) comprise between 10 and 20 SIBs, and only every eights workflow between 20 and 30. Only 9 workflows are composed of more than 50 SIBs. A similar distribution of workflow sizes has already been described by Littauer et al. in their paper "Trends in Use of Scientific Workflows: Insights from a Public Repository and Recommendations for Best Practices" [11], where they analyzed workflows stored in the myExperiment [5] public workflow repository. Indeed, we observed that most workflow designers develop their applications making use of several quite small submodels. This separates different levels of abstraction, prevents the models from becoming unmanageably large (which starts to be the case when the workflow model does not properly fit onto the modeling canvas any more), and results in small and easily reusable units of functionality.

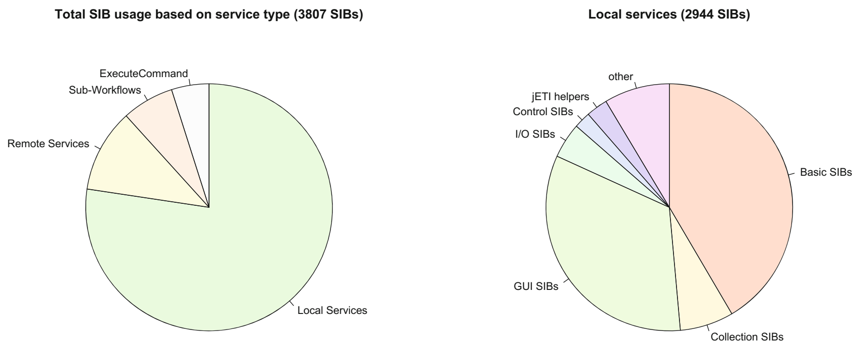


Fig. 11. Usage of different types of services

Figure 11 gives more information about the services that were used in the workflows. Following the service types distinguished by Wassink et al. in their study "Analysing Scientific Workflows: Why Workflows Not Only Connect Web Services" [21], the left pie chart visualizes the share of local services (detailed below), remote services (calls to Web services and jETI [14] services), sub-workflows (`GraphSIBs` and `MacroSIBs`) and `ExecuteCommand` SIBs (that call scripts or tools) in the total number of SIBs. In concert with their results, local services make up the largest part of SIBs, followed by the remote services. While we have then found a slightly bigger proportion of sub-workflows than of `ExecuteCommand` SIBs, Wassink et al. identified slightly more scripting tasks than sub-workflows in their sample.

The right pie chart in Figure 11 shows the composition of the local services used in the workflows. The largest part is made up by the `Basic SIBs`, which provide basic functionality for workflow modeling (such as basic data processing functions and control structures) and are part of the jABC's standard SIB library (the so-called `Common SIBs`). Second-largest part are the `GUI SIBs`, another part of the `Common SIBs`, which can be used for including dialogs for user interaction in the workflow. This complies with an observation that Garijo et al. made in their study "Common motifs in scientific workflows: An empirical analysis" [4], namely that steps for human interaction are increasingly used in scientific workflows. The third-largest share of local services used in the analyzed workflows are "other" SIBs that comprise different kinds of local services that are not part of any jABC standard library. Thus, these SIBs were primarily implemented for particular projects. The remaining parts of the pie chart are distributed between four other standard SIB libraries, namely the `Collection SIBs` and the `I/O SIBs` from the `Common SIBs` library, as well as the `Control SIBs` as part of the jABC framework and the `jETI helper SIBs` that come with the jETI remote execution plugin [14].

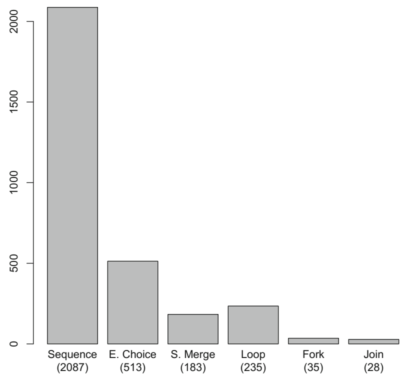


Fig. 12. Usage of workflow patterns

In order to learn more about the programmatic structure of scientific workflows, i.e., how the services are actually connected, we work on extending the process library to identify and count typical patterns that are used in scientific workflows (cf. Section 2.5). As a start, we focus on general, domain-independent workflow patterns as described by [1], which are particularly suitable to assess the control-flow and data-flow structures of a workflow. The current version of the corresponding analysis process is able to identify and count sequences, exclusive choices, simple merges, loops, forks, and joins. The detection of other patterns is the subject of future work. Figure 12 shows the preliminary results for our sample: The sequence pattern (simple sequential execution of two services) is by far the most-used pattern, with 2090 occurrences in the analyzed workflows. Exclusive choices (conditional branchings), simple merges (convergence of branches) and loops (repetitive behavior) also occur quite often, while parallel executions (fork/join) do in fact only play a minor role. As control-flow structures like conditional branchings and loops are apparently used very frequently, workflow systems should provide the possibility to include them in their workflows. While this is natural in control flow-based systems like the jABC, other mechanisms for their definition have to be developed for data flow-oriented systems.

4 Conclusion

We have presented a process library for the empirical analysis of jABC workflows that is itself composed of jABC workflow models and applied it to a selection of scientific workflows created by our students to generate first results. Note that in contrast to the analysis workflow described in [21], our workflow is completely modeled with the available standard SIB libraries of the jABC, that is, no additional scripts or services were implemented. In essence, our first results, covering workflow size and service usage, comply with the results obtained by previous studies on scientific workflows, such as [4,11,21]. As we have a large number of jABC workflows from a variety of application domains available, we are going to carry on this first study and analyze other sets of workflows as well. It will be especially interesting to investigate the differences between the various application domains, such as scientific and business processes and their different sub-domains.

In the context of empirical analysis of scientific workflows, it would be especially interesting to perform the analyses on the workflows in the myExperiment [5] workflow repository in order to compare the results for the jABC workflows with results for workflows from a data source that is often considered representative for the scientific community. Since the process library is currently limited to jABC workflows, it requires to extend the library to cover other modeling languages, which in particular means to extend those parts of the process library that deal with the processing of the concrete workflow models, which are of course tailored to the specific language. For the counting of used services (individually and in categories) the XML file in which the model is stored is analyzed. Since many workflow systems use some XML dialect for storing their workflows, it tends to be straightforward to adapt the process to be able to deal with other languages as well. For the analysis of the workflow patterns (structure of the model), an extension to other workflow languages requires more effort, since it is necessary that the workflows can be loaded and analyzed programmatically. Since the analyses are implemented for control-flow SLGs of the jABC framework, it will in particular be easier to transfer them to other control-flow models than to conceptually different data-flow models.

Currently, we are working on making the functionality of this the process library available as a jABC plugin, so that the analysis of workflows can easily be performed via the graphical user interface of this model-driven framework. As detailed in [18], plugin-based extensions are explicitly foreseen by the framework and it provides interfaces that greatly simplify their development and integration. Following the example of the PROPHETS process synthesis plugin [17], where users can define their own synthesis processes that are then executed by the plugin, we also plan to make it possible that users can define own analysis workflows. This would enable users to flexibly adapt the workflow analyses to their specific preferences and interests.

At the same time, we are exploring different extensions of the currently implemented functionality. It would for instance be desirable to generate the SIB statistics according to some domain-specific taxonomy (currently this is done according to the package hierarchy of the Java classes of the SIBs), to make use of the jABC's model checking plugin GEAR [2] for static analyses with regard to data-flow properties and program correctness, and to generate charts and other visualizations automatically. More advanced than the analysis of domain-independent workflow patterns but also by far more complex, would be the identification of domain-specific workflow patterns, as sketched in [7], that could be used for semantics-based workflow design support, for instance with the PROPHETS [10,17] process synthesis plugin.

References

1. van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: Workflow Patterns. Distributed and Parallel Databases 14(1), 5–51 (2003)
2. Bakera, M., Margaria, T., Renner, C., Steffen, B.: Tool-supported enhancement of diagnosis in model-driven verification. Innovations in Systems and Software Engineering 5, 211–228 (2009), `http://dx.doi.org/10.1007/s11334-009-0091-6`
3. Ebert, B.E., Lamprecht, A.L., Steffen, B., Blank, L.M.: Flux-P: Automating Metabolic Flux Analysis. Metabolites 2(4), 872–890 (2012), `http://www.mdpi.com/2218-1989/2/4/872`
4. Garijo, D., Alper, P., Belhajjame, K., Corcho, O., Gil, Y., Goble, C.: Common motifs in scientific workflows: An empirical analysis. Future Generation Computer Systems (2013) (in press), `http://www.sciencedirect.com/science/article/pii/S0167739X13001970`
5. Goble, C.A., Bhagat, J., Aleksejevs, S., Cruickshank, D., Michaelides, D., Newman, D., Borkum, M., Bechhofer, S., Roos, M., Li, P., Roure, D.D.: myExperiment: A repository and social network for the sharing of bioinformatics workflows. Nucleic Acids Research 38(suppl. 2), W677–W682 (2010), `http://nar.oxfordjournals.org/cgi/content/abstract/38/suppl_2/W677`
6. Jörges, S.: Construction and Evolution of Code Generators. LNCS, vol. 7747. Springer, Heidelberg (2013)
7. Lamprecht, A.-L. (ed.): User-Level Workflow Design. LNCS, vol. 8311. Springer, Heidelberg (2013)
8. Lamprecht, A.L., Margaria, T., Steffen, B.: Bio-jETI: A framework for semantics-based service composition. BMC Bioinformatics 10(suppl. 10), S8 (2009)
9. 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), `http://www.ncbi.nlm.nih.gov/pubmed/18460174`
10. Lamprecht, A.L., Naujokat, S., Margaria, T., Steffen, B.: Synthesis-Based Loose Programming. In: Proc. of the 7th Int. Conf. on the Quality of Information and Communications Technology (QUATIC 2010), Porto, Portugal, pp. 262–267 (September 2010)
11. Littauer, R., Ram, K., Ludäscher, B., Michener, W., Koskela, R.: Trends in Use of Scientific Workflows: Insights from a Public Repository and Recommendations for Best Practices. In: 7th International Digital Curation Conference (2011)
12. Margaria, T., Kubczak, C., Njoku, M., Steffen, B.: Model-based Design of Distributed Collaborative Bioinformatics Processes in the jABC. In: Proceedings of the 11th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 2006), pp. 169–176. IEEE Computer Society, Los Alamitos (2006)
13. Margaria, T., Kubczak, C., Steffen, B.: Bio-jETI: A service integration, design, and provisioning platform for orchestrated bioinformatics processes. BMC Bioinformatics 9(suppl. 4), S12 (2008)

14. Margaria, T., Nagel, R., Steffen, B.: jETI: A Tool for Remote Tool Integration. In: Halbwachs, N., Zuck, L.D. (eds.) TACAS 2005. LNCS, vol. 3440, pp. 557–562. Springer, Heidelberg (2005), http://www.springerlink.com/content/h9x6m1x21g5lknkx
15. Margaria, T., Steffen, B.: Agile IT: Thinking in User-Centric Models. In: Margaria, T., Steffen, B. (eds.) ISoLA 2008. CCIS, vol. 17, pp. 490–502. Springer, Heidelberg (2009)
16. Margaria, T., Steffen, B.: Service-Orientation: Conquering Complexity with XMDD. In: Hinchey, M., Coyle, L. (eds.) Conquering Complexity, pp. 217–236. Springer, London (2012), http://dx.doi.org/10.1007/978-1-4471-2297-5_10
17. Naujokat, S., Lamprecht, A.-L., Steffen, B.: Loose Programming with PROPHETS. In: de Lara, J., Zisman, A. (eds.) FASE 2012. LNCS, vol. 7212, pp. 94–98. Springer, Heidelberg (2012)
18. Naujokat, S., Neubauer, J., Lamprecht, A.L., Steffen, B., Jörges, S., Margaria, T.: Simplicity-First Model-Based Plug-In Development. In: Garbervetsky, D., Kim, S. (eds.) Special Issue of the 2nd International Workshop on Developing Tools as Plug-ins. Software: Practice and Experience. John Wiley & Sons, Ltd. (to appear)
19. Pohl, K., Böckle, G., van der Linden, F.J.: Software Product Line Engineering: Foundations, Principles and Techniques. Springer-Verlag New York, Inc., Secaucus (2005)
20. Steffen, B., Margaria, T., Nagel, R., Jörges, S., Kubczak, C.: Model-Driven Development with the jABC. In: Bin, E., Ziv, A., Ur, S. (eds.) HVC 2006. LNCS, vol. 4383, pp. 92–108. Springer, Heidelberg (2007), http://dx.doi.org/10.1007/978-3-540-70889-6_7
21. Wassink, I., van der Vet, P.E., Wolstencroft, K., Neerincx, P.B., Roos, M., Rauwerda, H., Breit, T.M.: Analysing Scientific Workflows: Why Workflows Not Only Connect Web Services. In: IEEE Congress on Services, pp. 314–321 (2009)

Automatic Annotation of Bioinformatics Workflows with Biomedical Ontologies

Beatriz García-Jiménez and Mark D. Wilkinson

Biological Informatics Group
Center for Plant Biotechnology and Genomics (CBGP), UPM - INIA
28223 Pozuelo de Alarcón (Madrid), Spain
beatriz.garcia@upm.es, markw@illuminae.com
http://www.wilkinsonlab.info

Abstract. Legacy scientific workflows, and the services within them, often present scarce and unstructured (i.e. textual) descriptions. This makes it difficult to find, share and reuse them, thus dramatically reducing their value to the community. This paper presents an approach to annotating workflows and their subcomponents with ontology terms, in an attempt to describe these artifacts in a structured way. Despite a dearth of even textual descriptions, we automatically annotated 530 myExperiment bioinformatics-related workflows, including more than 2600 workflow-associated services, with relevant ontological terms. Quantitative evaluation of the Information Content of these terms suggests that, in cases where annotation was possible at all, the annotation quality was comparable to manually curated bioinformatics resources.

Keywords: scientific workflows, web services, bioinformatics, semantic annotation, text mining, ontologies, tags, term extraction.

1 Introduction

As the demand grows for more transparent and reproducible scientific research [1], it becomes increasingly urgent to adopt more formal strategies for recording scientific methodology. The most common approach to such explicit process modelling takes the form of a scientific workflow. These digital artifacts formally describe the series of steps by which a scientific experiment was/will be conducted. Workflows may exist at a variety of levels of abstraction, ranging from general process overviews, to specific tools, the data-flow connections between them, and their associated execution parameters.

Formal workflows are generally authored and/or executed using purpose-built software, and a variety of design and enactment environments are used by e-scientists, such as Taverna [2], Kepler [3], Wings [4], and Vistrails [5]. These environments serve three main purposes: first, they attempt to generalize the interfaces between different workflow components (e.g. Web Services versus local command-line tools or scripts) so that they can be connected together without concern for the precise mechanism by which data will be passed between the components; second, they often offer a means to facilitate component discovery at design-time, either by menu-driven component selection [2], by contextually-aware suggestion [6], or by semi or fully automated construction [7] [4] to simplify the design process and/or reduce errors; finally, at enactment time, they mediate the data flow between components and, generally, capture additional information about the provenance of the workflow execution.

Apart from the key goal of enhancing the explicitness and transparency of scientific methodology, one of the most touted benefits of formal workflows is that they can, in principle, be shared, reused, and repurposed. With this goal, and in parallel with the increasing use of formal workflows in e-science, projects have emerged that aim to capture and publish these workflows for the purpose of rediscovery and reuse. The primary such repository in the Life Sciences is myExperiment [8], which archives workflows from most design and enactment environments.

To facilitate discovery, workflows submitted to myExperiment can be annotated with both a block of descriptive text, as well as free-text keywords or "tags"; however there is little to no control over the quality or quantity of these annotations. Task-appropriate workflow discovery, then, relies largely on the matching of keywords from within these freeform, sometimes very limited textual sources. Similarly, detailed comprehension of the functionality and suitability of a discovered workflow also depends largely on human interpretation of these textual annotations. For example, if the workflow requires edits for repurposing, deep examination of the individual workflow subcomponents is required in order to identify which portion of the workflow requires revision. Unfortunately, workflows are seldom, if ever, annotated at this level of granularity. As such, it becomes necessary to resolve the individual subcomponents to their own sources of documentation. Such documentation might be available, for example, within the WSDL document for a Web Service, or the record of that service in BioCatalogue, both of which are, again, either freeform tags or narrative text. While such traversals are plausible, there is currently no infrastructure that can reliably mechanize the traversal from a workflow subcomponent to its independent documentation. Moreover, the documentation of these subcomponents is as unregulated and often as sparse as that of the workflow itself, thus making them of dubious utility.

One approach to improving the status quo would be to semantically annotate both workflows and their subcomponents with ontological terms. Semantic annotations have numerous benefits over keyword and free-text annotations, such as supporting query expansion, filtering, precision, and computational tractability for formal verification and validation of workflow structures. There are currently no widely accepted standards for representing or annotating workflows though a variety of *de facto* standards are available; conversely, more widely accepted standards are available by which to capture semantic annotations for individual workflow components. For example, the World Wide Web consortium has recommended the SAWSDL standard for capturing ontological and controlled vocabulary terms within the structure of a traditional WSDL document representing a Web Service. This is intended to act as a bridge between traditional Web Services, and "Semantic Web Services", where each field in the conventional Web Service interface definition can now be mapped into an ontological context, together with an (optional) machine-readable data structure mapping, such as XSLT. Projects such as EMBRACE [9] are taking on the task of annotating legacy Web Services into SAWSDL using ontological terms from bioinformatics ontologies, in particular, EDAM (EMBRACE Data And Methods ontology [10]). Other projects aim to take advantage of even richer semantics. The SADI [11], SHARE [7] and Wings [4] projects all capture rich semantic annotations at the level of both the overall workflow, as well as the individual subcomponents, and hence, both support (to some degree) fully automated workflow assembly. In all of these cases, however, the semantic annotations are generated manually, either at the time of service/workflow authoring, or as part of a legacy migration and curation process. As such, it would be highly desirable to "boot-strap" the semantic annotation of legacy workflows and workflow subcomponents through some form of automated semantic annotation.

Here we describe an approach to the semantic annotation of legacy workflows in the myExperiment repository, as well as their component services. Workflows are first filtered to eliminate any steps that are exclusively syntactic transformations ("shims" [12]), with the resulting workflow skeleton containing only "biologically meaningful" operations. These skeletons are then mined using information from a variety of sources, including the myExperiment and BioCatalogue [13] repository entries, the BioMoby registry [14], and WSDL source documents. Mined descriptions are then processed to discover matches to nine relevant ontologies from the OBO Foundry. The resulting workflow templates are then reassembled using the representation of the Open Provenance Model for Workflows (OPMW) [15, 16] from the Wings project, which includes well-defined facets for capture of rich semantic annotations. Finally, we describe our degree of success in extracting such annotations, as well as attempting to quantitatively evaluate the quality of these annotations in terms of their Information Content [17].

2 Material and Methods

Section 2 describes the steps we undertook in our efforts to automatically annotate myExperiment workflows using terms from bioinformatics-relevant ontologies.

Overall, the system consumes Taverna workflows in Taverna 1 (*scufl*) or Taverna 2 (*t2flow*) formats [2], and outputs a set of ontology annotations linked to each available and 'biologically meaningful' service within each input workflow. As a secondary output, the partially-abstracted workflow (i.e. without data-type transformation nodes) is provided in *scufl* or *t2flow* format.

The following subsections describe the four primary phases of our analytical approach: 1) Filtering for bioinformatics-relevant workflows, 2) Cleaning "shim" services, 3) Retrieving service descriptions and 4) Entity extraction from descriptions to create the output semantic annotations.

2.1 Step 1: Filtering for Bioinformatics-Relevant Workflows

Our first requirement was to differentiate bioinformatics-oriented workflows from those relevant to other areas of investigation. As a first attempt, we selected workflows with the *'Bioinformatics'* tag; however, we observed that only 5% of Taverna workflows are described with this tag, leading to a high false-negative rate. We adjusted our criterion to search for specific bioinformatics-oriented topics, using relevant branches of EDAM [10] as our source vocabulary. Relevant EDAM terms were derived using the *'edamdef'* command from the EMBOSS package v6.4.0-4 [18], which allows us to search the definition of EDAM classes and returns terms matching the query term(s). The query:

```
edamdef -namespace topic -subclasses -query bioinformatics
```

'edamdef' returned 190 terms related to 'bioinformatics' from the 'topic' sub-ontology of EDAM. The description, title and tags of each myExperiment workflow were then searched using each of these terms, using the text mining Peregrine SKOS CLI software [19]. This filtering process resulted in 1206 presumptively bioinformatics-related workflows.

From manual inspection of these 1206 workflows, it became apparent that there were still an unacceptable number of false negatives because of the lack of an EDAM term in the description, title, or tags. Importantly, it was also apparent that some selected EDAM terms were too general, resulting in an unacceptable number of false positive workflows. For example, many false positives were discovered by matching the EDAM term 'workflows' (1022 cases, mixed with true positives) or 'ontologies' (29 cases).

After several iterations of trial and error together with manual verification of filtered and non-filtered workflows, we curated the list of filter terms, removing many of the most general EDAM classes which select workflows not specifically related to bioinformtics (e.g. ontologies, rna, structure, text mining, threading and workflows) and adding new specific terms to include bioinformatics workflows not retrieved with the EDAM classes (e.g. alignment, bioinformatics, BioMarker, bioMart, BioMoby, blast, chEBI, chemical, cheminformatics, EBI, ebi.ac.uk, ensembl, entrez, FASTA, GenBank, Gene expression, gene list, gene name, Gene Ontology, gene pattern, geneontology, genetic, genotyping, GO term, InterPro, Kegg, metagenomics, microarray, molecular, molecule, ncbi, openPHACTS, pathway, Pfam, phylogenetic, protein, PubMed, SNP, somatic, SwissProt, systems_biology uniprot, UniprotId and wikipathways). We then repeated the filtering process. Again, a manual examination of a subset of the filtered workflows suggested that approximately 95% of the erroneous filtering had been eliminated; the identified false positives had been eliminated, preserving the true positives, and many of the false negatives were now discovered. Remaining false negatives consisted of workflows with no description, no tags, no title and/or words not correctly space-separated —effectively, impossible to discover using our approach. We believe, however, that this set of terms provides sufficient filtering precision to be used in an automated annotation pipeline leading to a dataset of sufficiently high-quality to be used in downstream data mining.

At the end of this filtering phase, from an input of 1839 workflows, 775 workflows were determined to be relevant to bioinformatics, although just 739 workflows are available to download from myExperiment (4.65% not downloadable). Among the 739 available workflows, 272 of them are in Taverna 1 format (scufl) and 467 in Taverna 2 format (t2flow).

2.2 Step 2: Cleaning "shim" Services

Taverna workflows have been reported as containing many "shim" services [20] —that is, workflow elements that execute data transformations (merging, formatting, or parsing), but not biologically meaningful analyses [21]. These shims represent structural transformations, not biologically relevant transformations we are interested in. They do not contribute to our understanding of the science behind a workflow, and as such, we undertook to automatically identify and remove them from the workflow prior to the annotation phase of our analysis.

According to the Taverna User Manual[1], we considered as shim services the following categories: XML splitter, spreadsheet import, string/text constant, beanshell, local service and Xpath; and as non-shim services: WSDL, REST, bioMoby, bioMart, soaplab and Rshell. When a shim service is removed, the steps before and after that shim are reconnected in our dataset, thus preserving the "flow" of the workflow. Note, however, that the resulting T2flow files cannot be accurately visualized in Taverna (though they appear to be XML schema-compliant); nevertheless, since visualization was not our objective, nor was the objective to create a "runnable" workflow, this was not problematic for the remainder of our analysis. In some cases, the pruned workflow has services without inputs and/or outputs, and in other cases, pruned workflow is left with only inputs and outputs, if all its processors are shims. Figure 1 shows an example of a Taverna workflow before and after cleaning shims.

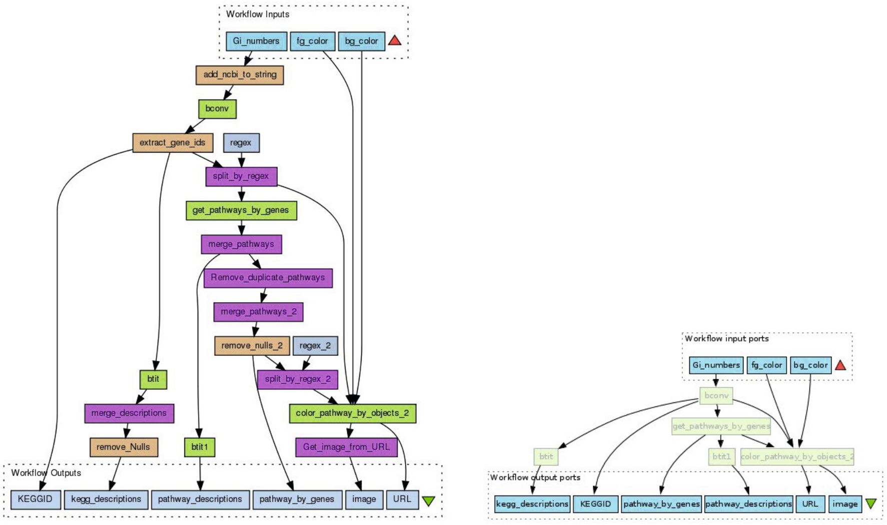
At the end of this second data preparation phase, we retain the same number of workflows overall, however each workflow now has fewer component processors. 77 workflows contained only shim services, and were therefore "empty" after this data preparation phase.

2.3 Step 3: Retrieving Service Descriptions

Here, we query myExperiment and a variety of service metadata repositories to obtain a textual description of each remaining service in each workflow. We focus our efforts on annotations present in WSDL, BioMoby, SoapLab, REST and nested workflows services, since they are the most frequent services in our workflows (see section 3.1) and have obvious metadata sources within which to search for annotations.

For each service, we attempt to construct a textual description that is composed of (if available): service name + service description + operation name + operation description. These four different elements are discovered from a variety

[1] `http://dev.mygrid.org.uk/wiki/display/taverna/Service+types`



(a) Before (b) After

Fig. 1. Example of workflow with cleaned shim services. (a) The original workflow is myExperiment workflow #1180, and (b) its associated service without shims.

of sources and sites, using several keys (e.g. endpoint URI or service name), with continuous checking for errors, and with multiple, possibly redundant attempts, attempting to locate the richest, most descriptive source possible until a description is found or all possibilities have been expended. The sources included any or all of: the myExperiment workflow entry; Scufl and T2flow files from myExperiment [8]; the WSDL source document for each service; the BioMoby registry entry [14][2]; and the BioCatalogue service repository [13] through its API to specific endpoint searches and general searches. For Scufl files, service descriptions were sometimes available for services within these files. T2Flow files do not have a descriptive field, other than for nested workflows, and as such it was always necessary to attempt retrieval of the WSDL source document, MOBY registry entry, or retrieve the relevant record from the BioCatalogue repository.

Disappointingly, very frequently, a dearth of annotations at the service level meant that the final textual description of a service was limited to just the service name (912 of 3560 bioinformatics services - 25.62%) with 246 services having no annotation whatsoever.

2.4 Step 4: Entity Extraction from Descriptions to Create Semantic Annotations

The final step in our annotation pipeline is to execute text analytics on the description from step 3, in order to ontologically annotate the services and,

[2] `http://moby.ucalgary.ca/cgi-bin/getServiceDescription`

subsequently, the workflow of which they are component. As such, the input to this step in the pipeline is a descriptive paragraph, and the output is a list of relevant ontology classes associated with that service description.

The ontologies that acted as the vocabulary source for the text analysis were: BioAssay Ontology (BAO), Bioinformatics Web Service Ontology (OBIWS), Biomedical Resource Ontology (BRO), EDAM Ontology of Bioinformatics Operations and Data Formats (EDAM), Experimental Factor Ontology (EFO), Information Artifact Ontology (IAO), Mass Spectrometry Ontology (MS), Medical Subject Headings (MESH), National Cancer Institute Thesaurus (NCIT), Neuroscience Information Framework Standard (NIFSTD), Ontology for Biomedical Investigations (OBI), Semanticscience Integrated Ontology (SIO) and Software Ontology (SWO).

These ontologies cover a variety of categories of concepts relevant to bioinformatic workflows, such as operations, topics, algorithms, etc. All of them must belong to the OBO Foundry [22] and BioPortal [23] library of ontologies, and thus can be used for annotation using the *Open Biomedical Annotator* [24]. The *Open Biomedical Annotator* is an application available from BioPortal [23], an open repository of commonly used biomedical ontologies and related tools. The *Open Biomedical Annotator* web service matched words in our descriptive paragraph to classes in selected ontologies by doing an exact string comparison (a 'direct' match) between our description and ontology class names, synonyms and identifiers.

In some cases, we noticed duplicated annotations (with the same or different URI), due to overlapping or explicit relations among different ontologies (e.g. SWO imports EDAM). In section 3.2 we present results referring to the total numbers of annotations before and after removing duplicated terms, where duplication is defined as sharing the same URI, but appearing in different ontologies. To remove the redundancy, we consider SWO, OBIWS, OBI, EFO and NIFSTD have preference to their imported ontologies. Terms with the same name/label, but differing URIs, are *not* considered to be identical, since that would require a deep, manual interpretation of the semantics of that term within each of the ontologies.

530 workflows from the 739 available bioinformatics workflows were successfully annotated with one or more ontological classes.

3 Results

First, this section presents a study of the workflows in terms of various categories of sub-component composition. Subsequently, we expose an analysis of the quantity and quality of the derived semantic annotations, based on a calculation of the Information Content represented by the set of semantic terms discovered. Finally, a sample of the final workflow annotations represented according to the OPMW model is provided.

3.1 Understanding Workflow Composition

Figure 2 illustrates the distribution of the different categories of services in all 739 bioinformatics workflows, split into shim and non-shim components. The most notable observation in Fig. 2 is that the number of shim elements far exceeds the number of biologically meaningful elements; more than 65% of all workflow components are shims (see Fig. 2(center)). This reinforces the importance of step 2 of our annotation system, but also highlights the complexity and penetrance of data transformation problems in bioinformatics, at least in part due to the proliferation of data formats, as has been argued for at least a decade [25].

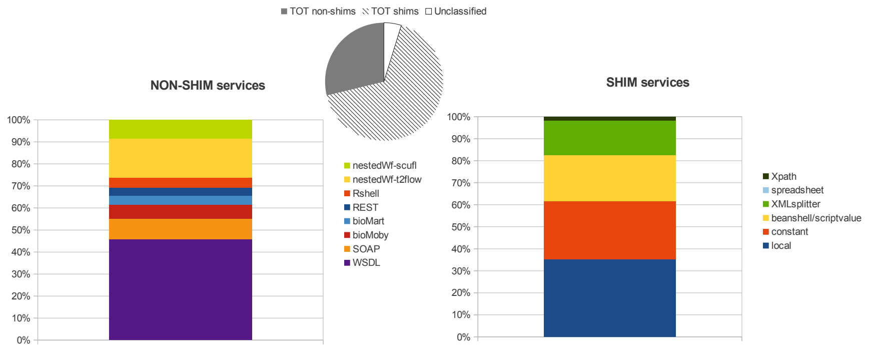


Fig. 2. Average distribution of shim and non-shims services. In the center, the global average distribution between shim and non-shims services in all bioinformatic workflows. On both sides, the average distribution of shim (right) and non-shim (left) categories of services.

Among the non-shim services (see Fig. 2(left)), the most frequent are WSDL services, where, together with SOAP and BioMoby services, these make-up 60% of all biologically-meaningful services. The most common categories of shim services (see Fig. 2(right)) are local, constant and beanshell/scriptvalue services, covering more than an 80% of all shims combined.

On average, each bioinformatic-related workflow has 16.74 components, where 11.13 are shims and 4.81 are non-shim services. The remaining 0.80 are other unclassified services. We observe that the ratio of shim/non-shim services is higher in T2flow format (12.50/4.81) than in Scufl format (8.77/4.83), where more than 2.5 shims are included per each domain service.

3.2 Annotation Analysis

In the final step of our analysis, we attempt to quantitatively evaluate the degree to which this annotation methodology yielded useful results.

One immediate consideration is that not all the services and workflows were amenable to automated annotation at all (i.e. had little or no information to mine for annotations). We consider this to be a failure of the scientific community, rather than a failure of our analysis, and as such, we take this into-account with respect to our final evaluation of the pipeline's success.

In total, we obtain 70636 ontological annotations spanning 2922 different ontological classes, of which 64324 are non-redundant (i.e. the identical URI appearing in multiple ontologies). This means we achieve at least one annotation for 2605 of 3560 non-shim services (73.17%). In terms of workflows, we collect the annotations of their instantiated services, without taking the order of the services into account. This allowed us to annotate 530 out of 739 workflows (71.72%) with at least one ontology class.

Clearly, not all semantic annotations are equally informative. For example, the ontology term "analysis" is less informative than the ontology term "fastq file parser". Thus, we attempted to measure the informativeness and the relative quality of our automatically-generated annotations. To achieve this, we apply a semantic metric based on Information Theory —the Information Content (IC) [17]— and we computed the IC value of each automatically-selected ontology term, each service, and each workflow.

We choose an intrinsic IC metric for a variety of reasons, such as the topology of the taxonomy, the lack of a "gold standard" annotated knowledgebase, and in order to avoid biases and the dependence on external annotations. Among the three available alternatives for intrinsic IC [17], we chose the Zhou et al. metric [26] which takes into account the number of descendants in a similar manner to that proposed by Seco et al. [27], but where the former approach also includes the depth of a term in the taxonomy. Although Sanchez et al. [17] improves this statistic by including the number of subsumers, we prefer Zhou et at. since they define a normalized metric, which therefore allows us to compare IC values of terms from distinct ontologies.

Using the Zhou et al. metric, we compute the IC values of each ontology class with the Semantic Measures Library Toolkit [28]. Thereafter, we describe the 'informativity' of the set of semantic annotations associated with a service as the IC for that service, computed as the best (the maximum) IC value of every ontology annotation associated to that service. This measure is independent of the redundancy present within the annotation terms, since it takes only the maximum for any given term. Finally, we defined the degree of informativity of the annotations of an entire workflow, as the IC per workflow, computed as the average of all the IC values of the services within the workflow. The top row in Fig. 3 shows the resulting IC value distribution of those automatic annotation. Note that only services that had some analysable description are included in these IC values computations, since the others could not be scored. Other reasons for not being scorable include annotation with obsolete terms, or terms not included in the main ontology hierarchy since they are external properties associated with a core ontology class, and therefore not part of the IC

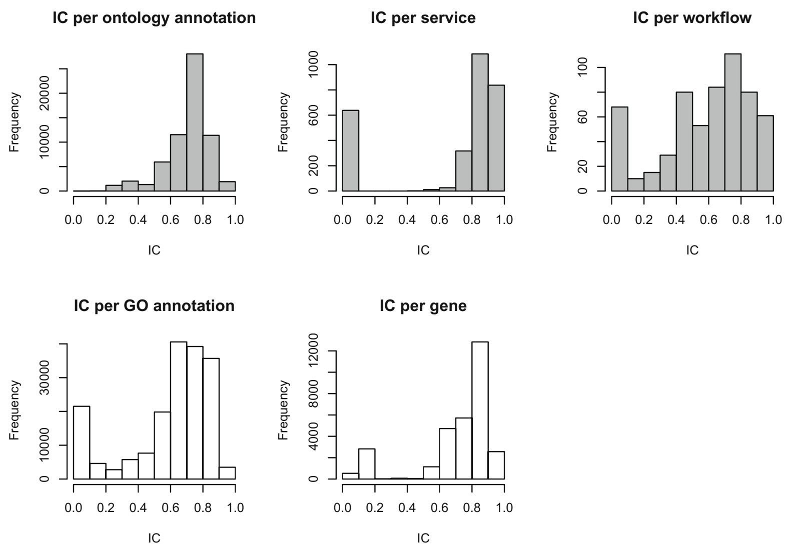


Fig. 3. Histograms of Information Content (IC) values associated to ontological annotations. Top row (grey columns) shows IC distribution of automated annotations generated by our system. Bottom row (white columns) shows IC distribution of manual GO annotations of *Arabidopsis Thaliana*. Frequency axes are not directly comparable.

statistic, such as terms from MESH qualifiers and MESH supplementary concept records.

Having selected an intrinsic metric for IC, we were consequently unable to objectively define what a "good" IC value would be. Moreover, we do not have access to a "gold standard" set of annotated bioinformatics services with which to compare our automated annotations. As such, we opted to execute our IC analysis on a set of ontological annotations done largely manually, and considered to be of high-quality by the community. In particular, we take the set of genes of *Arabidopsis thaliana* manually annotated with Gene Ontology (GO) terms from the TAIR FTP site [29]. We compute the IC values for each gene, as the IC of the set of GO terms associated with that gene, using the same procedure as with our automatic annotations; thus, we infer a hypothetical correspondence between an annotated locus, and an annotated service. The results of these IC values corresponding to manual annotations is shown in bottom row of Fig. 3.

When both rows are compared in Fig. 3, we observe a similar distribution of IC values resulting from our automated annotations (top row) compared to the (largely) manual annotations (bottom row), with the notable exception of the high number of services with annotation IC of 0 (due to a lack of annotations). On average, the IC value per automated annotation is 0.7139 versus 0.5986 for the manual GO annotations (first column) and the average IC value per annotated service is 0.8707 if all IC=0 services are excluded 0.6801 if they are included) versus 0.7172 of manual annotation per gene. We note that genes of unknown function (5229 Arabidopsis genes) are excluded from the GO annotation file, in a manner similar to our filtering-out of services with no annotation, increasing the validity of this comparison. Therefore, we could conclude that the informativeness, in terms of IC values, of our automatic annotations are as good or better than what we might expect from manual annotations.

To compare IC values split by ontology, we compute IC per ontology as the average of all the annotations within each ontology. We conclude SIO provides the best annotations (0.8172), followed by NCIT (0.7529) and SWO (0.7519); additionally, MESH has the highest minimum IC value (0.3459). In terms of quantity of annotations, NCIT is the best with 31853 annotations (1544 different terms). The ten most frequent annotations were (in descending order): *job resource* (NIFSTD), *computer job* (NCIT), *occupation* (NCIT), *gene/s* (MESH and NCIT), *protein* (EFO and NCIT) and *database* (NIFSTD, MESH and EDAM).

3.3 Workflow Annotations in OPMW Model

To make our annotations available in a structured and reusable way, we chose to represent them in RDF as instances of the *Workflow Template Process* class[3] from the Open Provenance Model for Workflows (OPMW) ontology [15, 16]. While it may appear to be more desirable to publish these as SAWSDL documents, as recommended by EMBRACE, we elected not to do so, since (a) not all of our annotated services are originally published as WSDL and, moreover (b) there is no obvious way to re-publish the derived SAWSDL files in a way that would be discoverable/usable by the community (i.e. they cannot be re-associated with their respective services or workflows in either the BioCatalogue or myExperiment repositories).

Figure 4 shows the OPMW structure that describes the semantic annotations of one service. Our RDF output files[4], includes an instance of this model for each non-shim available service of each annotated bioinformatics workflow. Using this RDF file, our annotations are available and could be easily integrated in other systems requiring structured annotations of bioinformatic services.

```
<SERVICE URL>
      a opmw:ProcessTemplate, <ontology class 1 URL>, <ontology class ...>, <ontology class N>;
      opmw:template <WORKFLOW URL>;         # link to workflow which this service is part of.
      opmw:uses <DATA URL>.                 # link to previous service in the workflow.
```

Fig. 4. Abstract structure to define our semantic annotations in one service

In addition to this primary output, we also provide the derived set of partially-abstracted workflows (i.e. after removing non-biologically-meaningful steps) in

[3] `http://www.opmw.org/ontology/WorkflowTemplateProcess`

[4] Available at `http://wilkinsonlab.info/myExperiment_Annotations/OPMW/*`

Taverna formats[5] with the caveat that these are for informational purposes only, and cannot be accurately visualized, nor run, in Taverna.

4 Discussion and Conclusions

The limitations of this approach to bootstrapping annotations are obvious (both *a priori* and as borne-out in the results). Namely, there was a well anticipated difficulty in finding descriptive annotations which could be mined for semantic meaning. Beyond that, however, the heterogeneity of the content and representation of the workflows also made it difficult to discover, mine, and even select appropriate ontologies for the annotation effort. We will now go into some details about how this affected the accuracy and/or comprehensiveness of each step in our annotation pipeline.

With respect to the first step —filtering for bioinformatics-related workflows— when a workflow is submitted to myExperiment, fields that could justifiably be considered "core metadata", such as description, and title, are not mandatory. The same can be said of service submission to the various Web Service registries. Disappointingly, workflow and service submitters therefore can, and too often do, opt to disregard these most basic metadata elements. Given that we depended on this metadata for our first-pass filter of relevance, we can be certain that our outcomes were adversely affected by these "nuisance-behaviours". In the absence of these basic information elements, it would be extremely difficult even for a human to determine the function and/or relevance of a workflow for a particular task, and it was clearly an insurmountable problem for an automated annotation pipeline. Finally, one additional limitation to our success in the first stage of our pipeline was that not all selected workflows were available to download. MyExperiment returned 'not found' or 'not authorized' errors in these cases, and thus these workflows had to be removed from our analysis.

Lack of descriptive annotations became acutely problematic in step 3 of our pipeline, when we attempted to construct a description of individual services within the workflow. It was frequently the case (25.62% of 'biologically relevant' services) that the files describing a service would have no description at all beyond the service name. We would suggest that such lax, nonchalant behaviours on the part of submitting scientists entirely defeats the purpose of submitting to a public repository. Therefore we believe that it would not be unreasonable for the various workflow and service registries to be more demanding of authors with regard to these fundamental annotations.

This same problem manifested itself in the semantic annotation retrieval from step 4, where the quantity and quality of the derived annotations depend, obviously, on the amount of descriptive text available; the longer the description, the more informative the annotations in most cases. For example, services with short descriptions (such as "prophet: Scan one or more sequences with a Gribskov or Henikoff profile", with 63 characters), result in few and very general ontology

[5] Available at `http://wilkinsonlab.info/myExperiment_Annotations/abstract_workflows/`

annotations (*Sequence analysis* (EDAM) and *scan* (MS), with a IC per service of 0.5785). While services with longer descriptions result often in many and specific ontology annotations (such as "Eigen_analysis", with 3946 characters, with 263 annotations, with an IC per service of 0.9588).

A distinct source of error arose from service deprecation. Several services referred-to in workflows had been deprecated, and the sources of documentation (if they ever existed) were absent. Although through manual exploration we determined that some of these deprecated services have been replaced by new services, and these new services had annotations in Biocatalogue, it was very difficult to automatically discover when such deprecation/replacement had taken place based on the reference to that service in the original workflow, and it was not clear how to automate this complex traversal. It seems, therefore, that some clearly-defined method for tracking versioning is required for workflow sub-components, and that this tracking mechanism should have features that allow it to be automated.

Finally, existing annotations were missed due to external repositories' errors and bad (malformed or inappropriate) responses to search queries. These cases, though representing only a fraction of the entries, were the result of either unstable interfaces and/or errors (or non-documented limitations) in the various APIs.

Finally, related to the quality of the automated annotations, we emphasise that IC value can only give a measure of the informativeness of the annotations; it cannot report on their appropriateness vis-à-vis the real function of the service or workflow. Moreover, IC value is determined largely by the topology of the ontology, and because ontologies vary in their granularity from branch-to-branch, high IC values do not necessarily imply "rich" annotations. For example, some annotations with high IC values, such as *patient, synonym, human* and *length*, with IC scores greater than 0.95, do not seem, subjectively, particularly informative in the context of the aims of our study. However, IC is an objective and useful measure when an ontology is applied to annotate a knowledge base [30] and we could not identify an objective alternative.

Altogether, we feel there is significant room for improvement in the straightforward capture of core, non-semantic metadata at the point of resource submission. We demonstrate here how this, in turn, could allow automated semantic annotations of surprisingly high quality to be extracted via text mining approaches. Such semantic annotations could then be used to improve the submission process itself, by for example, detecting when certain important types of metadata are missing, and/or prompting for likely annotations based on existing patterns detected by Machine Learning techniques.

Acknowledgments. MDW is funded by the Isaac Peral/Marie Curie Cofund programme (FP7) of the European Union and UPM; BGJ is funded by the Isaac Peral Programme of UPM.

References

1. Micheel, C.M., Nass, S.J., Omenn, G.S. (eds.): Evolution of Translational Omics Lessons Learned and the Path Forward. The Institute of Medicine of the National Academies (2012)
2. Oinn, T., Addis, M., Ferris, J., et al.: Taverna: A tool for the composition and enactment of bioinformatics workflows. Bioinformatics 20(17), 3045–3054 (2004)
3. Altintas, I., Berkley, C., Jaeger, E., et al.: Kepler: An extensible system for design and execution of scientific workflows. In: Proceedings of the 16th International Conference on Scientific and Statistical Database Management, pp. 423–424 (2004)
4. Gil, Y., Ratnakar, V., Kim, J., et al.: Wings: Intelligent Workflow-Based Design of Computational Experiments. IEEE Intelligent Systems (2011)
5. Callahan, S.P., Freire, J., Santos, E., et al.: Vistrails: visualization meets data management. In: Proceedings of the 2006 ACM SIGMOD, pp. 745–747. ACM (2006)
6. Withers, D., Kawas, E., McCarthy, L., Vandervalk, B., Wilkinson, M.: Semantically-Guided Workflow Construction in Taverna: The SADI and BioMoby Plug-Ins. In: Margaria, T., Steffen, B. (eds.) ISoLA 2010, Part I. LNCS, vol. 6415, pp. 301–312. Springer, Heidelberg (2010)
7. 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)
8. Goble, C.A., Bhagat, J., Aleksejevs, S., et al.: myExperiment: A repository and social network for the sharing of bioinformatics workflows. Nucleic Acids Research 38(suppl. 2), W677–W682 (2010)
9. Rice, P.M., Bleasby, A.J., Haider, S.A., et al.: EMBRACE: Bioinformatics data and analysis tool services for e-Science. In: Second IEEE International Conference on e-Science 2006, p. 146 (2006)
10. Ison, J., Kala, M., Jonassen, I., et al.: EDAM: An ontology of bioinformatics operations, types of data and identifiers, topics and formats. Bioinformatics 29(10), 1325–1332 (2013)
11. Wilkinson, M.D., Vandervalk, B., McCarthy, L.: The Semantic Automated Discovery and Integration (SADI) Web service Design-Pattern, API and Reference Implementation. Journal of Biomedical Semantics 2(1), 8 (2011)
12. Radetzki, U., Leser, U., Schulze-Rauschenbach, S.C., et al.: Adapters, shims, and glue–service interoperability for in silico experiments. Bioinformatics 22(9), 1137–1143 (2006)
13. Bhagat, J., Tanoh, F., Nzuobontane, E., et al.: BioCatalogue: A universal catalogue of web services for the life sciences. Nucleic Acids Research (May 2010)
14. Wilkinson, M.D., Senger, M., Kawas, E., et al.: Interoperability with Moby 1.0–it's better than sharing your toothbrush! Briefings in Bioinformatics 9(3), 220–231 (2008)
15. Garijo, D., Gil, Y.: A new approach for publishing workflows: Abstractions, standards, and linked data. In: Proceedings of the WORKS 2011, Held in Conjunction with SC 2011, Seattle, Washington, pp. 47–56. ACM (2011)
16. Garijo, D., Gil, Y.: Towards open publication of reusable scientific workflows: Abstractions, standards, and linked data. Technical report (January 2012)
17. Sáchez, D., Batet, M., Isern, D.: Ontology-based information content computation. Knowledge-Based Systems 24(2), 297–303 (2011)
18. Rice, P., Longden, I., Bleasby, A.: EMBOSS: The European Molecular Biology Open Software Suite. Trends in Genetics 16(6), 276–277 (2000)

19. Schuemie, M., Jelier, R., Kors, J.: Peregrine: Lightweight gene name normalization by dictionary lookup. Peregrine CLI SKOS. In: Proceedings of the Biocreative 2 Workshop, Madrid, April 23-25 (2007), `https://trac.nbic.nl/biosemantics/wiki/PeregrineSKOSCLI`
20. Garijo, D., Alper, P., Belhajjame, K., et al.: Common motifs in scientific workflows: An empirical analysis. In: 8th IEEE International Conference on eScience (2012)
21. Wolstencroft, K., Haines, R., Fellows, D., et al.: The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud. Nucleic Acids Research 41(W1), W557–W561 (2013)
22. Smith, B., Ashburner, M., Rosse, C., et al.: The OBO Foundry: coordinated evolution of ontologies to support biomedical data integration. Nature Biotechnology 25(11), 1251–1255 (2007)
23. Whetzel, P.L., Noy, N.F., Shah, N.H., et al.: BioPortal: Enhanced functionality via new Web services from the National Center for Biomedical Ontology to access and use ontologies in software applications. Nucleic Acids Research 39(suppl. 2), W541–W545
24. Shah, N., Bhatia, N., Jonquet, C., et al.: Comparison of concept recognizers for building the Open Biomedical Annotator. BMC Bioinformatics 10(suppl. 9) (2009)
25. 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)
26. Zhou, Z., Wang, Y., Gu, J.: A new model of information content for semantic similarity in wordnet. In: Second International Conference on FGCNS 2008, vol. 3, pp. 85–89 (December 2008)
27. Seco, N., Veale, T., Hayes, J.: An intrinsic information content metric for semantic similarity in wordnet. In: ECAI 2004, including PAIS 2004, pp. 1089–1090 (2004)
28. Harispe, S., Ranwez, S., Janaqi, S., et al.: The semantic measures library and toolkit: fast computation of semantic similarity and relatedness using biomedical ontologies. Bioinformatics 30(5), 740–742 (2014)
29. Lamesch, P., Berardini, T.Z., Li, D., et al.: The Arabidopsis Information Resource (TAIR): improved gene annotation and new tools. Nucleic Acids Research 40(D1), D1202–D1210 (2012)
30. Good, B.M.: Strategies for amassing, characterizing, and applying third-party metadata in bioinformatics. PhD thesis, University of British Columbia (2009)

Evaluation and Reproducibility of Program Analysis (Track Introduction)

Markus Schordan[1], Welf Löwe[2], and Dirk Beyer[3]

[1] Center for Applied Scientific Computing
Lawrence Livermore National Laboratory, CA, USA
schordan1@llnl.gov
[2] Department of Computer Science
Linnaeus University, Sweden
welf.lowe@lnu.se
[3] Faculty of Computer Science and Mathematics
University of Passau, Germany

Track Description

Today's popular languages have a large number of different language constructs and standardized library interfaces. The number is further increasing with every new language standard. Most published analyses therefore focus on a subset of such languages or define a language with a few essential constructs of interest. More recently, program-analysis competitions [4,6,7,1] aim to evaluate comparatively implemented analyses for a given set of benchmarks. The comparison of the analyses focuses on various aspects: (a) the quality of established structures and automata describing the behavior of the analyzed program, (b) the verification of various specified program properties, (c) the impact on a client analysis of particular interest, and (d) the impact of analysis precision on program optimizations.

This track is concerned with the methods of comparative evaluation of program analyses and how analysis results can be represented such that they remain reproducible and reusable as intermediate results for other analyses. We therefore focus on how analysis results can be specified and how to allow an exact re-computation of the analysis results irrespectively of a chosen (internal) intermediate representation. This includes specification languages for program properties and program-analysis results, its representation in existing analysis infrastructures, compilers, and tools, along with meta-models and evolution of these representations. It also requires to address the reuse of verification results, the combination of multiple verifiers using conditional model checking [2], and how to overcome obstacles in combining tools that implement different approaches (e.g., model checking and data-flow analysis). To further fuel discussion of the above topics, this track also includes a panel, in which the various presented approaches and tools are discussed interactively.

Contributions

Björn Lisper presents SWEET [8], a tool for worst-case execution time (WCET) analysis. It aims to estimate the longest possible execution time for a piece of code executing uninterrupted on a particular hardware. The tool combines a number of different analyses that have been developed over the years. SWEET establishes flow facts about a program for computing an estimate of the execution time. The portability of the analyses is addressed by operating on a representation of the input program in the open intermediate format ALF [5].

George Chatzieleftheriou, Apostolos Chatzopoulos, and Panagiotis Katsaros present a methodology for systematically evaluating and comparing analysis tools [3]. The methodology aims to systematically vary analysis requirements in order to detect code defects, and assess a static-analysis tool's effectiveness in a wide range of potential coding complexities. Following this methodology, an extensive evaluation is performed on a publicly available test suite consisting of 750 programs for 30 distinct code defects. The investigated code defects for C programs include integer overflows, truncation errors, format-string vulnerabilities, memory leaks, absence of failure checks, and deadlocks in concurrent programs. Since the evaluation can be repeated at any time, it also allows to investigate and assess the improvements of released versions of a given tool.

In the last decades, several branches of the static analysis of imperative programs have made significant progress, such as in the inference of numeric invariants or the computation of data-structure properties (using pointer abstractions or shape analyzers). Although simultaneous inference of invariants of the shape of dynamic data structures and the numeric values stored in theses memory cells is often needed, the case of combing both is especially challenging and less well explored. Notably, simultaneous shape-numeric inference raises complex issues in the design of the static analyzer itself. Xavier Rival, Antoine Toubhans, and Bor-Yuh Evan Chang present an approach for the modular construction of static analyzers with abstract domains for heterogeneous properties [10]. It is based on the combination of multiple atomic abstract domains to describe several kinds of memory as well as value properties.

Markus Schordan, Pei-Hung Lin, Dan Quinlan, and Louis-Nol Pouchet present an approach for verifying polyhedral optimizations of programs with floating-point operations [11]. The presented approach is independent from floating-point precision and therefore applies to a large number of optimization variants, independent from a respective platform and details in modeling floating-point precision. By combining partial evaluation, a rewrite system, and matching, the semantic equivalence of the original and the optimized program is verified for various polyhedral optimization variants. More than 1000 variants are verified for the benchmarks in the publicly available Polybench/C 3.2 benchmark suite [1].

In a world that increasingly relies on the Internet to function, application developers rely on the implementations of protocols to guarantee the security of data transferred. Whether a chosen protocol gives the required guarantees, and whether the implementation does the same, is usually unclear. Jose Quaresma, Christian W. Probst, and Flemming Nielson present the guided system-development framework [9], which aims at making development and verification of secure

[1] http://www.cs.ucla.edu/~pouchet/software/polybench/

communication systems easier by bridging the gap between system development and verification of communication protocols.

The combination of various areas of program analysis that are presented in this track, from timing analysis to abstract domains for heterogeneous properties, verification of floating-point computations to protocols, development of secure communication systems, and extensive evaluations of program-analysis tools, forms a rich basis for the discussion of aspects of evaluation and reproducibility. Together, these contributions provide an overview of the comprehensive response of the research community to the increasing importance of analysis and verification in the design and development of evolving software.

References

1. Beyer, D.: Status report on software verification (competition summary SV-COMP 2014). In: Ábrahám, E., Havelund, K. (eds.) TACAS 2014. LNCS, vol. 8413, pp. 373–388. Springer, Heidelberg (2014)
2. Beyer, D., Henzinger, T.A., Keremoglu, M.E., Wendler, P.: Conditional model checking: A technique to pass information between verifiers. In: Bultan, T., Robillard, M. (eds.) Proc. FSE. ACM (2012)
3. Chatzieleftheriou, G., Chatzopoulos, A., Katsaros, P.: Test-driving static analysis tools in search of C code vulnerabilities II (Extended abstract). In: Margaria, T., Steffen, B. (eds.) ISoLA 2014, Part II. LNCS, vol. 8803, pp. 486–488. Springer, Heidelberg (2014)
4. Combe, D., de la Higuera, C., Janodet, J.-C.: Zulu: An interactive learning competition. In: Yli-Jyrä, A., Kornai, A., Sakarovitch, J., Watson, B. (eds.) FSMNLP 2009. LNCS (LNAI), vol. 6062, pp. 139–146. Springer, Heidelberg (2010)
5. Gustafsson, J., Ermedahl, A., Lisper, B., Sandberg, C., Källberg, L.: Alf: A language for WCET flow analysis. In: OASIcs-OpenAccess Series in Informatics, vol. 10. Schloss Dagstuhl-Leibniz-Zentrum für Informatik (2009)
6. Howar, F., Isberner, M., Merten, M., Steffen, B., Beyer, D.: The RERS Grey-Box Challenge 2012: Analysis of event-condition-action systems. In: Margaria, T., Steffen, B. (eds.) ISoLA 2012, Part I. LNCS, vol. 7609, pp. 608–614. Springer, Heidelberg (2012)
7. Howar, F., Isberner, M., Merten, M., Steffen, B., Beyer, D., Pasareanu, C.S.: Rigorous Examination of Reactive Systems. The RERS Challenges 2012 and 2013. In: Software Tools for Technology Transfer (2014)
8. Lisper, B.: SWEET – A tool for WCET flow analysis (Extended abstract). In: Margaria, T., Steffen, B. (eds.) ISoLA 2014, Part II. LNCS, vol. 8803, pp. 482–485. Springer, Heidelberg (2014)
9. Quaresma, J., Probst, C.W., Nielson, F.: The Guided System Development Framework: Modeling and verifying communication systems. In: Margaria, T., Steffen, B. (eds.) ISoLA 2014, Part II. LNCS, vol. 8803, pp. 509–523. Springer, Heidelberg (2014)
10. Rival, X., Toubhans, A., Chang, B.-Y.E.: Construction of abstract domains for heterogeneous properties (Position paper). In: Margaria, T., Steffen, B. (eds.) ISoLA 2014, Part II. LNCS, vol. 8803, pp. 489–492. Springer, Heidelberg (2014)
11. Schordan, M., Lin, P.-H., Quinlan, D., Pouchet, L.-N.: Verification of polyhedral optimizations with constant loop bounds in finite state space computations. In: Margaria, T., Steffen, B. (eds.) ISoLA 2014, Part II. LNCS, vol. 8803, pp. 493–508. Springer, Heidelberg (2014)

SWEET – A Tool for WCET Flow Analysis (Extended Abstract)

Björn Lisper

School of Innovation, Design, and Engineering, Mälardalen University,
SE-721 23 Västerås, Sweden

1 Introduction

Worst-Case Execution Time (WCET) analysis [14] aims to estimate the longest possible execution time for a piece of code executing uninterrupted on a particular hardware. Such WCET estimates are used when analysing real-time systems with respect to possible deadline violations. For safety-critical real-time systems, *safe* (surely not underestimating) estimates are desirable. Such estimates can be produced by a *static WCET analysis* that takes all possible execution paths and corresponding hardware states into account.

Static WCET analysis has been around for 20 years, and a number of tools have emerged such as aiT [5], Otawa [2], Bound-T [8], Heptane [3], TuBound [11], and Chronos [10]. Most tools today use the so-called "Implicit Path Enumeration Technique" (IPET) [12]. In IPET, execution times are estimated from local WCET bounds for small program fragments (typically basic blocks). Each such fragment p is given an *execution counter* $\#p$ recording its number of executions: the execution time for a path is then approximated from above by the sum $\sum_p \mathit{WCET}(p) \times \#p$, where $\mathit{WCET}(p)$ is the local WCET bound for p. WCET estimation can now be formulated as maximising this sum subject to *program flow constraints* on the execution counters. If these constraints are linear, then the WCET estimation becomes an Integer Linear Programming (ILP) problem that can be solved by a standard ILP solver. It turns out that very many important program flow constraints can be expressed as linear constraints.

Thus, in the IPET model the WCET estimation problem is nicely decomposed into three distinct parts: the *low-level analysis*, which computes local WCETs using hardware timing models, the *program flow analysis* that derives program flow constraints ("Flow Facts") from the program code, and the final *calculation* where the ILP problem is solved to produce the WCET bound. Notably the program flow analysis will not need any information about hardware timing, but can be based entirely on the functional semantics of the code. Flow Facts can indeed be seen as a special kind of loop invariants.

2 SWEET

SWEET (SWEdish Execution Time tool) is a tool that derives Flow Facts automatically. SWEET can compute a variety of Flow Facts, from simple loop

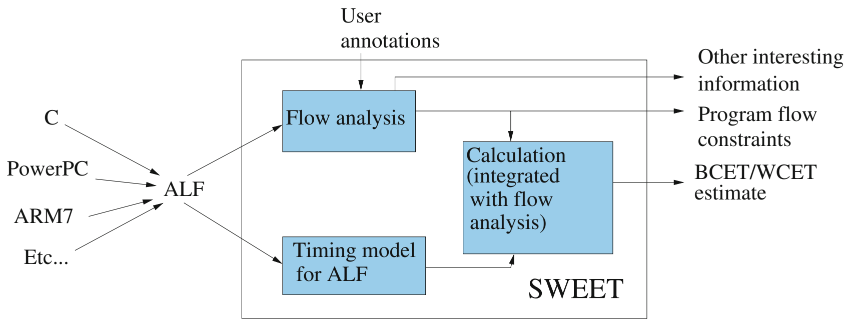


Fig. 1. The structure of SWEET

iteration bounds to complex infeasible path constraints. It can analyze a variety of code formats through translation into the intermediate format ALF [6]. SWEET is open source: comprehensive information about the tool is found online [13]. In Fig. 1 the structure of SWEET is shown.

An earlier version of SWEET ("NIC-SWEET") was integrated into a research compiler, and could analyze code generated by that compiler. This version of SWEET was a full WCET analysis tool using the IPET model. The current version ("ALF-SWEET") is a specialised program flow analysis tool.

SWEET uses *Abstract Execution* (AE) [7] to derive Flow Facts. AE can be seen as a very context-sensitive value analysis, where different loop iterations are analysed separately. This gives the analysis a flavor of symbolic execution, executing the program in the abstract domain using abstract states rather than concrete states. AE is based on the theory of abstract interpretation: thus it is safe, and computed Flow Facts will never underestimate the set of possible program paths. SWEET currently uses an abstract domain of bounded intervals, but AE also works with other abstract domains.

Abstract states reaching a condition may contain concrete states for which the condition evaluates to true and false, respectively. Then the abstract state is split into a different abstract state for each outcome of the condition. To curb the potential explosion of states SWEET offers a variety of *merge strategies*, where abstract states are merged in certain program points using their least upper bound operator. By selecting the proper strategy, the tradeoff between precision and analysis speed can be fine-tuned.

AE is a potentially very general technique to derive Flow Facts. It can in principle deal with loops of any form, as long as the abstract domain can express the semantics of the loop conditions accurately enough. AE can also bound recursion depth. The price to pay for this generality is a risk of nontermination. The current implementation in SWEET has some limitations: recursion is not handled, as well some forms of unstructured loops. The use of interval domain also yields some limitations. SWEET currently handles nontermination by allowing the user to set a timeout where the analysis is aborted.

The environment of the analysed code may be important to know for the analysis. For instance, the values of some variables may be confined to certain ranges. SWEET provides *abstract input annotations*, where such constraints can be specified. The AE can use this information to compute tighter Flow Facts.

SWEET uses *recorders* and *collectors* to compute Flow Facts during the AE [7]. Recorders are attached to abstract states, and contain information that is successively accumulated into the collectors during the abstract execution. Collectors are pertinent to *scopes* (typically loops), and their final values are used to produce Flow Facts for that scope. For instance, to compute an upper loop bound the recorder is the execution counter for the loop header, and the collector is a number containing the highest value of this counter seen for any abstract state in the loop so far. Other, more complex Flow Facts are generated using more elaborate recorders and collectors.

SWEET can compute different kinds of Flow Facts specified by a combination of attributes telling the type of bound (upper/lower/infeasible), where to put execution counters, and Flow Fact context. The Flow Facts can thus be context sensitive (call strings), and they can pertain to different scopes (e.g., an execution counter for the loop body in a nested loop can be relative to either the inner or outer loop). SWEET has an expressive language for expressing these Flow Facts. In addition, SWEET can generate Flow Facts in the annotation formats of the commercial WCET analysis tools aiT and RapiTime.

In order to keep SWEET portable across different formats it analyses the intermediate format ALF [6]. Other languages and formats can be analysed if translated into ALF. To facilitate this, ALF is designed to faithfully represent high-level languages (like C) as well as machine code. Currently two translators from C to ALF exist, as well as a translator from PowerPC binaries to ALF.

The current version of SWEET lacks a low-level analysis. It can however use simple timing models for ALF to obtain WCET estimates. This estimation is done directly in the AE by treating time as a variable being incremented for each executed statement [4]. The AE thus computes an interval bounding the execution time. This interval also bounds the execution time from below, thus providing a Best Case Execution Time (BCET) estimate. Such simple cost models are way to coarse to provide both safe and tight WCET/BCET bounds, but they can nevertheless be useful to provide approximate bounds, for instance for early source-level timing estimation [1].

SWEET can also provide information from its rich set of internal analyses supporting the AE. Such analyses include a conventional value analysis, data flow analysis, construction of control flow and call graphs, and program slicing.

3 Conclusions

We have presented SWEET, a tool for generating precise Flow Facts. It is designed for maximal interoperability. It can be used both as a standalone analysis tool, or as a "plugin" providing Flow Facts to other tools: indeed, SWEET is an important component in the Open Timing Analysis Platform [9]. Its main use is however as a vehicle for program analysis research targeting real-time code.

References

1. Altenbernd, P., Ermedahl, A., Lisper, B., Gustafsson, J.: Automatic generation of timing models for timing analysis of high-level code. In: Faucou, S. (ed.) Proc. 19th International Conference on Real-Time and Network Systems (RTNS 2011), Nantes, France (Sepember 2011)
2. Ballabriga, C., Cassé, H., Rochange, C., Sainrat, P.: OTAWA: An open toolbox for adaptive WCET analysis. In: Min, S.L., Pettit, R., Puschner, P., Ungerer, T. (eds.) SEUS 2010. LNCS, vol. 6399, pp. 35–46. Springer, Heidelberg (2010)
3. Colin, A., Puaut, I.: A modular and retargetable framework for tree-based WCET analysis. In: Proc. 13th Euromicro Conference on Real-Time Systems (ECRTS 2001) (June 2001)
4. Ermedahl, A., Gustafsson, J., Lisper, B.: Deriving WCET bounds by abstract execution. In: Healy, C. (ed.) Proc. 11th International Workshop on Worst-Case Execution Time Analysis (WCET 2011), Porto, Portugal (July 2011)
5. Ferdinand, C., Heckmann, R., Franzen, B.: Static memory and timing analysis of embedded systems code. In: 3rd European Symposium on Verification and Validation of Software Systems (VVSS 2007), Eindhoven, The Netherlands, pp. 153–163. No. 07-04 in TUE Computer Science Reports (March 2007)
6. Gustafsson, J., Ermedahl, A., Lisper, B., Sandberg, C., Källberg, L.: ALF – a language for WCET flow analysis. In: Holsti, N. (ed.) Proc. 9th International Workshop on Worst-Case Execution Time Analysis (WCET 2009), pp. 1–11. OCG, Dublin (2009)
7. Gustafsson, J., Ermedahl, A., Sandberg, C., Lisper, B.: Automatic derivation of loop bounds and infeasible paths for WCET analysis using abstract execution. In: Proc. 27th IEEE Real-Time Systems Symposium (RTSS 2006), pp. 57–66. IEEE Computer Society, Rio de Janeiro (2006)
8. Holsti, N., Saarinen, S.: Status of the Bound-T WCET tool. In: Proc. 2nd International Workshop on Worst-Case Execution Time Analysis, WCET 2002 (2002)
9. Huber, B., Puffitsch, W., Puschner, P.: Towards an open timing analysis platform. In: Healy, C. (ed.) Proc. 11th International Workshop on Worst-Case Execution Time Analysis (WCET 2011), Porto, Portugal (July 2011)
10. Li, X., Liang, Y., Mitra, T., Roychoudhury, A.: Chronos: A timing analyzer for embedded software. Science of Computer Programming 69(1-3), 56–67 (2007)
11. Prantl, A., Schordan, M., Knoop, J.: TuBound – a conceptually new tool for worst-case execution time analysis. In: Kirner, R. (ed.) Proc. 8th International Workshop on Worst-Case Execution Time Analysis (WCET 2008), Prague, Czech Republic, pp. 141–148 (July 2008)
12. Puschner, P.P., Schedl, A.V.: Computing maximum task execution times – a graph-based approach. Journal of Real-Time Systems 13(1), 67–91 (1997)
13. SWEET home page (2011), `http://www.mrtc.mdh.se/projects/wcet/sweet/`
14. Wilhelm, R., Engblom, J., Ermedahl, A., Holsti, N., Thesing, S., Whalley, D., Bernat, G., Ferdinand, C., Heckmann, R., Mitra, T., Mueller, F., Puaut, I., Puschner, P., Staschulat, J., Stenström, P.: The worst-case execution time problem — overview of methods and survey of tools. ACM Transactions on Embedded Computing Systems (TECS) 7(3), 1–53 (2008)

Test-Driving Static Analysis Tools in Search of C Code Vulnerabilities II
(Extended Abstract)

George Chatzieleftheriou, Apostolos Chatzopoulos, and Panagiotis Katsaros

Department of Informatics, Aristotle University of Thessaloniki
54124 Thessaloniki, Greece
{gchatzie,aachatzop,katsaros}@csd.auth.gr

Keywords: static analysis, software security, benchmark tests.

A large number of tools that automate the process of finding errors in programs has recently emerged in the software development community. Many of them use static analysis as the main method for analyzing and capturing faults in the source code. Static analysis is deployed as an approximation of the programs' runtime behavior with inherent limitations regarding its ability to detect actual code errors. It belongs to the class of computational problems which are undecidable [2]. For any such analysis, the major issues are: (1) the programming language of the source code where the analysis is applied (2) the type of errors to be detected (3) the effectiveness of the analysis and (4) the efficiency of the analysis.

In order to incorporate a static analysis tool for detecting potential code defects in the software development cycle, significant costs are required. Thus, it is important to know if such a tool is effective in finding all types of errors and especially the critical ones for ensuring product quality. It is also a matter of major importance to know how efficient a tool is with respect to the size of the code bases being analyzed. When two or more static analysis tools are compared based on code bases from existing software projects the results are biased: they actually refer to the tools' capability to detect only the defects within the tested code bases, which are characterized by a specific degree of program complexity and size. We believe that empirical studies on open source programs should be completed with evaluation results, which cover systematically the most frequent code defects in a specific software context.

The main focus of our work [1] is on software security and reliability. We have created a versatile test suite, which implements code defects for the C programming language. Our test suite is based on those errors which are more often reported in public catalogs. We have identified major defect categories, such that all examined defects are classified in one of them. Category *General* includes three types of flaws, namely division by zero, use of uninitialized variables and null pointer dereference. The second category, *Integers*, includes integer overflows, sign and truncations errors. Direct overflows, off-by-one errors and unbounded copies appear in categories *Arrays* and *Strings* along with the format string vulnerabilities [8], string truncation and null termination errors.

Many frequent C code defects are presented in category *Memory*, such as double free attempts, improperly allocated memory, initialization errors, memory leaks, absence of failure checks and access in previously freed memory. Category *File operation* contains the errors of redundant file closure, omission of file closure, absence of failure check and access in a file that is either, previously closed, not opened or opened with a different mode. Last but not least, category *Concurrency* errors includes deadlocks and time-of-check-time-of-use (TOCTOU) errors.

Our methodology aims to systematically vary analysis requirements in order to detect the mentioned code defects, and assess the static analysis tool effectiveness in a wide range of potential coding complexities. Our publicly available test suite consists of 750 programs for 30 distinct code defects from the mentioned categories. All programs include one line with the tested flaw and another line of code used to check the tools' capability to avoid reporting spurious errors. The test suite was applied to four open-source [3] [4] [5] [6], and two commercial tools [7], whose effectiveness was measured using metrics such as accuracy, precision, recall, specificity and F-measure. Accuracy is the ratio of correct classifications over the total number of observations. Precision is the ratio of the number of true positives over the number of reported errors. Recall is the ratio of the number of true positives over the number of actual errors. Specificity is the ratio of the number of true negatives over the sum of true negatives and false positives. The F-measure provides an aggregate measure for precision and recall, two metrics that are characterized by an intrinsic tradeoff. We also measured the tools' efficiency in terms of running time and peak memory usage.

We have evaluated the tools' effectiveness based on a wide range of C constructs and different conditions of language semantics under which the defects may arise. Each defect is reproduced in many different programs, which are used to assess the default configuration of the static analysis tools with respect to their path sensitivity, context sensitivity and alias analysis capabilities. The test programs for analyzing the tools' efficiency were automatically generated such that for each case of different program size between 1000 and 7000 lines of code, three programs with different analysis sensitivity requirements are considered, namely path sensitivity, context-sensitivity and alias analysis.

The main outcome from test driving the referenced static analysis tools showed that only one open-source tool can compete and in fact was found superior over the commercial ones, in terms of precision. On the other hand, the tested commercial tools had a higher recall compared to all tested open source tools. This finding shows that their analyses are designed and configured, such that they are able to detect as many defects as possible with slightly lower precision than the tool described in [6]. However, the higher precision of the open-source tool is accompanied by a significant cost in analysis efficiency: for test programs with 7000 lines of code the average analysis running time was more than two times the average running time of the commercial tools.

Our methodology can be easily extended towards diverse quality contexts and software domains, and can be enriched for tool comparisons for other programming languages. As an interesting scenario, we consider its application for validating runtime safety of applications for a mobile computing platform. Such a type of validation is often a formal requirement for the distribution of applications through internet-wide markets and the procedure usually requires certification based on platform-specific security needs.

The results show how the evaluated tools compete in terms of important tradeoffs between analysis effectiveness and efficiency, as well as between precision and recall. The degree of extensibility and customization that each tool offers to the user should also be taken into account. In the last few years, the theory and the technology of static program analysis is rapidly developed and the market's driving forces call for new ways to balance the discussed tradeoffs between analysis effectiveness and efficiency. For this reason, we believe that there is an undeniable need to regularly repeat and publish every few years systematic studies such the one reported in [1].

Acknowledgment. This research has been co-financed by the European Union (European Social Fund ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) - Research Funding Program: Thalis Athens University of Economics and Business - SOFTWARE ENGINEERING RESEARCH PLATFORM.

References

1. Chatzieleftheriou, G., Katsaros, P.: Test-Driving Static Analysis Tools in Search of C Code vulnerabilities. In: Proc. of the 2011 IEEE 35th Annual Computer Software and Applications Conference Workshops, COMPSACW 2011 (2011)
2. Landi, W.: Undecidability of static analysis. ACM Lett. Program. Lang. Syst. 1(4), 323–337 (1992)
3. Evans, D., Larochelle, D.: Improving Security Using Extensible Lightweight Static Analysis. IEEE Softw. 19(1), 42–51 (2002)
4. Holzmann, G.J.: Static source code checking for user-defined properties. In: Proc. IDPT, vol. 2 (2002)
5. Cppcheck - A Tool for static C/C++ static code analysis, `http://sourceforge.net/apps/mediawiki/cppcheck`
6. Cuoq, P., Kirchner, F., Kosmatov, N., Prevosto, V., Signoles, J., Yakobowski, B.: Frama-C: A software analysis perspective. In: Eleftherakis, G., Hinchey, M., Holcombe, M. (eds.) SEFM 2012. LNCS, vol. 7504, pp. 233–247. Springer, Heidelberg (2012)
7. Parasoft C++ Test, `http://www.parasoft.com/`
8. One, A.: Smashing the stack for fun and profit. Phrack Magazine 7(49), 14–16 (1996)

Construction of Abstract Domains for Heterogeneous Properties (Position Paper)*

Xavier Rival[1], Antoine Toubhans[1], and Bor-Yuh Evan Chang[2]

[1] INRIA, ENS, CNRS, Paris, France
[2] University of Colorado, Boulder, Colorado, USA
{rival,toubhans}@di.ens.fr, bec@cs.colorado.edu

Abstract. The aim of static analysis is to infer invariants about programs that are tight enough to establish semantic properties, like the absence of run-time errors. In the last decades, several branches of the static analysis of imperative programs have made significant progress, such as in the inference of numeric invariants or the computation of data structures properties (using pointer abstractions or shape analyzers). Although simultaneous inference of shape-numeric invariants is often needed, this case is especially challenging and less well explored. Notably, simultaneous shape-numeric inference raises complex issues in the design of the static analyzer itself. We study the modular construction of static analyzers, based on combinations of atomic abstract domains to describe several kinds of memory properties and value properties.

Static Analysis to Infer Heterogeneous Properties. Static analysis by abstract interpretation [4] utilizes an *abstraction* to over-approximate (non-computable) sets of program states, using computer-representable elements, that stand for *logical properties* of concrete program states. As an example, for numerical properties, the interval abstract domain [4] uses constraints of the form $n \leq x$ and $x \leq p$ to describe possible values of variable x, where n, p are scalars.

To construct a static analyzer capable of inferring sound approximations of program behaviors, one designs an *abstract domain*, which consists of an abstraction, and abstract operations for sound post-condition operators, join and widening:

1. An abstraction is defined by a set of abstract elements $\mathbf{A}$ and a concretization function $\gamma : \mathbf{A} \to \mathcal{P}(C)$, which maps each abstract property $\mathbf{a}$ into the set of concrete elements $\gamma(\mathbf{a})$ that satisfy it. The set $\mathbf{A}$ of abstract elements will be assumed to be defined by a grammar of admissible logical predicates (e.g., for intervals, $\mathbf{a}(\in \mathbf{A}) ::= \mathbf{a} \wedge \mathbf{a} \mid n \leq x \mid x \leq p$).
2. A post-condition operator is a function $\mathbf{f} : \mathbf{A} \to \mathbf{A}$ which over-approximates a concrete operation $f : C \to \mathcal{P}(C)$ encountered in programs (as, e.g., a test).
3. Abstract join computes an over approximation of union and widening [4] enforces the termination of abstract iterates for the analysis of loops.

* The research leading to these results has received funding from the European Research Council under the FP7 grant agreement 278673, Project MemCAD and the United States National Science Foundation under grant CCF-1055066.

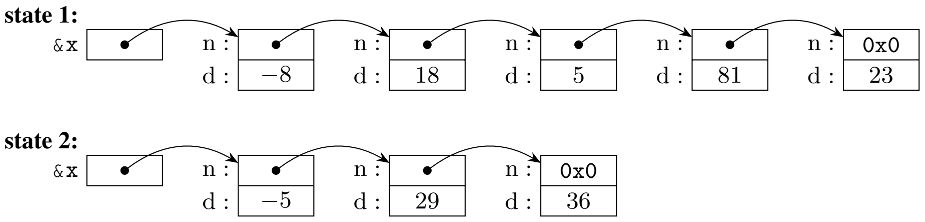


Fig. 1. Heterogeneous property abstraction

The combination of post-condition operators and widening operators allows us to define a sound static analyzer [4].

In the following, we discuss the design of an abstraction able to handle heterogeneous properties, about both data-structures and values. For instance, Figure 1 shows a couple of concrete states containing lists of numbers that are all positive except for the first one, which belongs to interval $[-10, 0]$: our goal is to engineer abstract domains able to express such properties, yet can be applied to many static analysis problems.

Abstraction of dynamic memory properties. For instance, a memory abstract domain consists of a set of predicates describing memory regions, together with operators for the analysis of memory operations (look-ups, assignments) and widening. XISA [3,2] relies on points-to predicates, inductive predicates and segment predicates. A simplified version of this abstraction, where the only inductive predicates and segments that are considered are lists boils down to the following:

$$\begin{array}{rll}
& \text{symbolic variables } \alpha, \alpha', \ldots & \text{denote values and addresses} \\
\mathbf{m}(\in \mathbf{M}) ::= & \mathbf{m} * \mathbf{m} & \text{separating conjunction of predicates} \\
| & \alpha \cdot \mathrm{f} \mapsto \alpha' & \text{cell field f at address } \alpha \text{ containing value } \alpha' \\
| & \mathrm{list}(\alpha) & \text{a list at address } \alpha \\
| & \mathrm{list}(\alpha') \Rightarrow\!\!* \ \mathrm{list}(\alpha) & \text{a list segment starting at } \alpha \text{ and ending at } \alpha'
\end{array}$$

The XISA [3] implementation actually represents a larger set of predicates, with arbitrary inductive definitions (including trees, doubly-linked lists and others). Other analysis frameworks utilize other sets of logical properties, such as, e.g., TVLA [9], which is based on reachability predicates.

Adding tracking for value properties, and departing from monolithic abstract domains. Once an abstraction has been defined for memory states, it is natural to extend it with value properties, so as to let the analysis infer constraints over both the structure of data and their values. A straightforward way to achieve this, and to add interval constraints over values is to extend the definition of abstract elements by $\mathbf{m} ::= \ldots \mid \mathbf{m} \wedge \alpha \leq n \mid \mathbf{m} \wedge n \leq \alpha \mid \ldots$. However, this implies the abstract operations (post-condition operators, join and widening) have to be extended so as to deal with both structures and value properties, at the same time: therefore abstract operations are bound to become overly complex. Moreover, this approach is awkward, as it does not build upon existing abstract operations of value abstractions such as intervals [4] or octagons [8], which means it will not easily benefit from the efficient algorithms designed to infer such properties (the same also applies to the memory abstraction). Besides, it makes it harder to switch from one value abstraction to another at a later point, hence reducing the flexibility of the analysis.

In the following, we advocate a *modular abstract domain design*, which:

- separates concerns in the abstract domain designs;
- reuses existing abstract domains algorithms;
- allows one to tune distinct parts of the abstractions independently.

Such design has been extensively used in the ASTRÉE static analyzer [1], which makes intensive use of reduced product [5] among other abstract domain combination techniques [6]. This design contributed not only to the precision and efficiency of the analysis, but also to making it easier to extend [6].

Abstraction of value properties, and combined abstract domain. To achieve a modular abstract domain design, we set up a different abstract domain $\mathbf{V}$ that will only track value properties (and not memory layout as the previously defined $\mathbf{M}$ does), and define a new abstract domain $\mathbf{S}$ for states that combines both:

$$\begin{array}{rll} \mathbf{m}(\in \mathbf{M}) & ::= \dots & \text{defined as before} \\ \mathbf{v}(\in \mathbf{V}) & ::= \mathbf{true} \mid \mathbf{v} \wedge \mathbf{v} \mid \alpha \leq n \mid n \leq \alpha & \text{value predicates} \\ \mathbf{s}(\in \mathbf{S}) & ::= \mathbf{m} \wedge \mathbf{v} & \text{conjunction of sub-properties} \end{array}$$

In essence, $\mathbf{S}$ defines a *reduced product* [5] of the memory abstraction $\mathbf{M}$ and value abstraction $\mathbf{V}$. As such, it completely separates memory and value abstraction concerns, which makes the abstract domain fully modular [11]. Indeed, both sub-components can be implemented in distinct ML modules, and $\mathbf{S}$ is defined as a ML functor. In practice, this functor should ensure that the symbolic variables used in the value abstraction are consistent with the memory cell contents and addresses symbols defined in the memory abstraction (thus it implements a co-fibered abstract domain [12], which essentially generalizes the notion of reduced product).

Both concrete states of Figure 1 can be abstracted by $\alpha \mapsto \alpha_0 * \alpha_0 \cdot \mathrm{n} \mapsto \alpha_1 * \alpha_0 \cdot \mathrm{d} \mapsto \alpha_0' * \mathrm{lpos}(\alpha_1) \wedge \alpha = \&\mathrm{x} \wedge -10 \leq \alpha_0' \wedge \alpha_0' \leq 0$, where inductive definition lpos describes all lists of positive numbers.

Separate combination of memory abstractions. So far, we combined abstract domains capturing distinct sets of properties. Yet, this abstract domain decomposition approach can be pushed further. As an example, ASTRÉE [1] relies on a decomposition of the numerical abstract domain into simpler abstractions that handle specific sets of properties. Likewise, a similar approach can be applied to the memory abstraction part. One approach to do this is to split concrete heaps and apply distinct memory abstractions to *disjoint* regions [11]:

$$\begin{array}{rll} \mathbf{m}(\in \mathbf{M}) & ::= \mathbf{m_0} * \mathbf{m_1} & \text{where } \mathbf{m_0} \in \mathbf{M_0} \wedge \mathbf{m_1} \in \mathbf{M_1} \\ \mathbf{m_0}(\in \mathbf{M_0}) & ::= \dots & \text{defines a 1st memory abstract domain, e.g., for lists} \\ \mathbf{m_1}(\in \mathbf{M_1}) & ::= \dots & \text{defines a 2nd memory abstract domain, e.g., for arrays} \end{array}$$

This construction allows one to apply parsimoniously expensive memory abstractions to the memory regions that require them, while lighter weight abstractions can be used for simpler structures. This results in better control of the analysis complexity. A cost is that the analyzer now has to resolve memory fragments across sub-domains, and to also select which memory fragment is the most adequate to account for each memory allocation.

Reduced product of memory abstractions. Likewise, one can design a reduced product [5] of memory abstract domains [10]:

$$\mathbf{m}(\in \mathbf{M}) ::= \mathbf{m_0} \wedge \mathbf{m_1} \qquad \text{where } \mathbf{m_0} \in \mathbf{M_0} \wedge \mathbf{m_1} \in \mathbf{M_1}$$

Such a composed abstraction is adequate when considering *overlaid* data structures [7] (such as lists or trees of objects with a common field pointing to class methods) and separates the concerns of analyzing each aspects of the structures. In turn, it imposes on the analysis the burden to let logical predicates represented in one sub-domain be usable to refine the computations done in the other sub-domain.

Modular abstract domain design. A modular abstract domain significantly simplifies the design of static analyzers while offering additional flexibility and control. The cost for this benefit is the innovation needed to design these more complex and general abstract domain combinators, but this cost is quickly amortized with the ability to reuse these combinators to realize arbitrary static analyzer configurations.

References

1. Blanchet, B., Cousot, P., Cousot, R., Feret, J., Mauborgne, L., Miné, A., Monniaux, D., Rival, X.: A static analyzer for large safety-critical software. In: PLDI (2003)
2. Chang, B.-Y.E., Rival, X.: Relational inductive shape analysis. In: POPL (2008)
3. Chang, B.-Y.E., Rival, X., Necula, G.C.: Shape analysis with structural invariant checkers. In: Riis Nielson, H., Filé, G. (eds.) SAS 2007. LNCS, vol. 4634, pp. 384–401. Springer, Heidelberg (2007)
4. Cousot, P., Cousot, R.: Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: POPL (1977)
5. Cousot, P., Cousot, R.: Systematic design of program analysis frameworks. In: POPL (1979)
6. Cousot, P., Cousot, R., Feret, J., Mauborgne, L., Miné, A., Monniaux, D., Rival, X.: Combination of abstractions in the astrée static analyzer. In: Okada, M., Satoh, I. (eds.) ASIAN 2006. LNCS, vol. 4435, pp. 272–300. Springer, Heidelberg (2008)
7. Lee, O., Yang, H., Petersen, R.: Program analysis for overlaid data structures. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 592–608. Springer, Heidelberg (2011)
8. Miné, A.: The octagon abstract domain. HOSC 19(1), 31–100 (2006)
9. Sagiv, M., Reps, T.W., Wilhelm, R.: Solving shape-analysis problems in languages with destructive updating. In: POPL (1996)
10. Toubhans, A., Chang, B.-Y.E., Rival, X.: Reduced product combination of abstract domains for shapes. In: Giacobazzi, R., Berdine, J., Mastroeni, I. (eds.) VMCAI 2013. LNCS, vol. 7737, pp. 375–395. Springer, Heidelberg (2013)
11. Toubhans, A., Chang, B.-Y.E., Rival, X.: An abstract domain combinator for separately conjoining memory abstractions. In: Müller-Olm, M., Seidl, H. (eds.) SAS 2014. LNCS, vol. 8723, pp. 285–301. Springer, Heidelberg (2014)
12. Venet, A.: Abstract cofibered domains: Application to the alias analysis of untyped programs. In: Cousot, R., Schmidt, D.A. (eds.) SAS 1996. LNCS, vol. 1145, pp. 366–382. Springer, Heidelberg (1996)

Verification of Polyhedral Optimizations with Constant Loop Bounds in Finite State Space Computations

Markus Schordan[1], Pei-Hung Lin[1], Dan Quinlan[1], and Louis-Noël Pouchet[2]

[1] Lawrence Livermore National Laboratory
{schordan1,lin32,dquinlan}@llnl.gov
[2] University of California Los Angeles
pouchet@cs.ucla.edu

Abstract. As processors gain in complexity and heterogeneity, compilers are asked to perform program transformations of ever-increasing complexity to effectively map an input program to the target hardware. It is critical to develop methods and tools to automatically assert the correctness of programs generated by such modern optimizing compilers.

We present a framework to verify if two programs (one possibly being a transformed variant of the other) are semantically equivalent. We focus on scientific kernels and a state-of-the-art polyhedral compiler implemented in ROSE. We check the correctness of a set of polyhedral transformations by combining the computation of a state transition graph with a rewrite system to transform floating point computations and array update operations of one program such that we can match them as terms with those of the other program. We demonstrate our approach on a collection of benchmarks from the PolyBench/C suite.

1 Introduction

The hardware trend for the foreseeable future is clear: symmetric parallelism such as in SIMD units is ubiquitous; heterogeneous hardware exemplified by System-on-Chips becomes the solution of choice for low-power computing; and processors' instruction sets keep growing with specialized instructions to leverage additional acceleration/DSP hardware introduced by manufacturers. This ever-increasing complexity of the computing devices is exacerbating the challenge of programming them: to properly harness the potential of a given processor, one has to significantly transform/rewrite an input program to match the features of the target hardware. Advanced program transformations such as coarse-grain parallelization, vector/SIMD parallelization, data locality optimizations, etc. are required to achieve good performance on a particular hardware. Aggressive optimizing compilers, as exemplified by *polyhedral compilation* [1], aim at automating these transformation stages to deliver a high-performance program that is transformed for a particular hardware. From a single input source, these compilers perform highly complex loop transformations to expose the proper granularity of parallelism and data locality needed for a given processor. Such transformations include complex loop tiling and coarse-grain parallelization.

Polyhedral compilers have shown great promises in delivering high-performance for a variety of targets from a single input source (for example to map affine stencil computations for CPUs [2,3], GPUs [4] and FPGAs [5]), where each target requires its dedicated set of program transformations. However, asserting the correctness of the generated code has become a daunting task. For example, on a 2D Finite-Difference Time-Domain kernel, after transformation the loop bound expressions of the only parallel OpenMP `for` loop generated are about 15 lines long, making manual inspection out of reach. We also remark that verifying the polyhedral compiler engine itself, PoCC [6], which is the result of 8 years of multi-institution development is also out of reach: the compiler is actually around 0.5 million lines of code, making the effort of producing a certification of these compiler optimizations in a manner similar to Leroy's Compcert work [7] extremely high.

In addition to high-level program transformations that are performed by the compiler, a series of low-level implementation choices can significantly challenge the design of a verification system. A compelling example relates to the floating point number implementation chosen by the back-end compiler. If one program is implemented using `double` precision (e.g., 64 bits) and the other program is implemented using for instance specialized 80 bits instructions, even if they are two totally equivalent programs in terms of semantics the output produced by these programs is likely to differ slightly: successive rounding and truncation effects will affect the output result, this even if the program is fully IEEE compliant.

We are in need for an automated system that asserts the correctness of the generated code by such optimizing compilers, in a manner that is robust to the back-end implementation decisions made by the compiler. Previous work such as Verdoolaege's [8] has focused on determining if two programs, in particular two *affine* programs [9], are semantically equivalent. Such tools require the input program (control flow and data flow) to be precisely modeled using combinations of affine forms of the surrounding loop iterators and program parameters. In contrast, our work uses a more practical approach with strong potential to be generalized to larger classes of programs. In the present work we focus on equivalence of affine programs where the value of all program parameters (e.g., problem size) is known at compile-time, with only simple data-dependent conditionals.

In this work, we propose an automated system to assert the equivalence of two affine programs, one of them being generated by the PolyOpt/C[1] compiler. At a high level, we combine the computation of a state transition graph with a rewrite system to transform the floating point operations and array update operations of one program such that we can match them (as terms) with those of the other program. We make the following contributions.

- We develop a new approach for determining the equivalence of two affine programs with simple data-dependent control-flow, leveraging properties of polyhedral optimizations to design a simple but effective rewriting system and equivalence checker.

- We provide extensive analysis of our method in terms of problem sizes and equivalence checking time.
- We evaluate our method on a relevant subset of polyhedral optimizations, asserting the correctness of PolyOpt/C on a collection of numerical kernels. Our work led to finding one bug in PolyOpt/C, which was not caught with its current correctness checking test suite.

The rest of the paper is organized as follows. Sec. 2 describes polyhedral transformations and the class of programs analyzed in this paper. Sec. 3 describes our equivalence checking method. Sec. 4 provides extensive evaluation of our approach, asserting the correctness of PolyOpt/C for the tested benchmarks. Sec. 5 discusses related work, before concluding.

2 Polyhedral Program Transformations

Unlike the internal representation that uses abstract syntax trees (AST) found in conventional compilers, polyhedral compiler frameworks use an internal representation of imperfectly nested affine loop computations and their data dependence information as a collection of parametric polyhedra, this enables a powerful and expressive mathematical framework to be applied in performing various data flow analysis and code transformations. Significant benefits over conventional AST representations of computations include the effective handling of symbolic loop bounds and array index functions, the uniform treatment of perfectly nested versus imperfectly nested loops, the ability to view the selection of an arbitrarily complex sequence of loop transformations as a single optimization problem, the automatic generation of tiled code for non-rectangular imperfectly nested loops [10,2,11], the ability to perform instancewise dataflow analysis and determine the legality of a program transformation using exclusively algebraic operations (e.g., polyhedron emptiness test) [9,12], and more [13]. The *polyhedral model* is a flexible and expressive representation for loop nests with statically predictable control flow. Loop nests amenable to this algebraic representation are called *static control parts* (SCoP) [9,13], roughly defined as a set of consecutive statements such that loop bounds and conditionals involved are affine functions of the enclosing loop iterators and variables that are constant during the SCoP execution (whose values are unknown at compile-time, a.k.a. program parameters). Numerous scientific kernels exhibit those properties; they can be found in image processing filters, linear algebra computations, etc.as exemplified by the PolyBench/C test suite [14].

In a polyhedral compiler, program transformations are expressed as a reordering of each *dynamic* instance of each syntactic statement in the program. The validity of this reordering is determined in PolyOpt/C by ensuring that the order in which each operation is accessing the same array cell is preserved in the transformed code, this follows the usual definition of data dependence preserving transformations [15]. No transformation on the actual mathematical operations used during the computation is ever performed: each statement body has its structure and arithmetic operations fully preserved after loop transformations.

Strength reduction, partial redundancy elimination, and other optimizations that can alter the statement body are not considered in the traditional parallelization/tiling polyhedral transformations [2] that we evaluate in PolyOpt/C. These properties allow the design of an analysis and a simple but effective rewriting rule system to proof equivalence, as shown in later Sec. 3.

In this work, we focus exclusively on polyhedral program variants that are generated by a polyhedral compiler, PolyOpt/C. Details on the variants considered are found in later Sec. 4. That is, the codes we consider for equivalence checking are *affine programs* that can be handled by PolyOpt/C. Technically, we consider a useful subset of affine programs where the loop bounds are fully computable at compile-time. That is, once the program has been transformed by PolyOpt/C (possibly containing program parameters, such as the problem/array sizes), the resulting program must have all parameters replaced by a numerical value, to ensure that loop bound expressions can be properly computed and analyzed by our framework. Looking at PolyBench/C benchmarks, a sample dataset is always provided, which implies that we know, at compile time, the value of all program parameters.

3 Approach

In our approach we verify that the sequence of update operations involving floating point operations on each array element in the original program is exactly the same as in an optimized version of the program. Our verification is a combination of static analysis, program rewrite operations, and a comparison based on an SSA form [16] where each array element is represented as a different variable (with its own SSA number).

Algorithm 1. DetermineFloatingPointAssignmentSequenceInSSA

```
Data: $P$ : Program
Result: $S_{ssa}$: sequence of floating point operations in SSA Form
$STG$=compute-STG($P$);
$A$=extract-floating-point-assignment-sequence($STG$);
foreach $a \in A$ do
    rewrite(a)          – apply rewrite rules 1-10
$S_{ssa}$=determineSSA(A);
```

In Algorithm 2 we use Algorithm 1 to determine the sequence of update operations for each program. Algorithm 1 first computes (statically) a state transition graph (STG). In the STG each node represents the state before an assignment or a condition. Edges represent state transitions. In our benchmark programs the loops have constant numeric bounds. We can therefore compute in each state a concrete value of each iteration variable. Floating point operations and updates on arrays are not evaluated. Next the (non-evaluated) operations on floating point variables are collected as a sequence of terms (function extract-array-element-assignment-sequence). We then apply 10 rewrite rules to normalize

Algorithm 2. Verify

```
Data: $P_1, P_2$ : Programs to verify
Result: result: true when Programs can be determined to be equivalent,
        otherwise false
$S_1$=DetermineFloatingPointAssignmentSequenceInSSA($P_1$);
$S_2$=DetermineFloatingPointAssignmentSequenceInSSA($P_2$);
$S'_1$=sort($S_1$);      – sort by unique lhs SSA variable of assignment
$S'_2$=sort($S_2$);      – sort by unique lhs SSA variable of assignment
if match($S'_1$,$S'_2$) then
    return true;
else
    return false;
```

all extracted array updates. We remark that the `foreach` loop in Algorithm 1 is actually a parallel loop: each term rewriting can be computed independently of the others. On the normalized sequence of assignments we then determine an SSA Form.

Algorithm 2 matches the determined (normalized) floating point update sequences. Because these sequences are in SSA form, we can reorder them for the purpose of comparison. We sort each sequence and match the two sorted sequences of terms representing the assignments. If both sequences are equal, then the programs have been verified to perform an identical sequence of updates on floating point values. In the following sections we discuss each operation in detail.

3.1 Example

As running example we use the (smallest) benchmark Jacobi-1d-Imper. The original loop body is shown in Fig. 1 and an optimized variant ("tile_8_1_1" variant) is shown in Fig. 2. In the following sections we describe how to analyze and transform both programs to automatically verify that both programs are equivalent and the optimization performed by polyOpt/C is indeed semantics preserving and correct.

3.2 State Transition Graph Analysis

```
#pragma scop
for (t = 0; t < 2; t++) {
  for (i = 1; i < 16 - 1; i++)
    B[i] = 0.33333 * (A[i-1] + A[i] + A[i + 1]);
  for (j = 1; j < 16 - 1; j++)
    A[j] = B[j];
}
#pragma endscop
```

Fig. 1. Original Jacobi-1d-Imper benchmark (only the loop is shown). The variable t is used for computing the number of steps and is set to 2 in the experiments. Array size is 16.

```
#pragma scop
{
  int c0;
  int c2;
  for (c0 = 1; c0 <= 17; c0++) {
    if (c0 >= 15) {
      if ((c0 + 1) % 2 == 0) {
        A[14] = B[14];
      }
    }
    for (c2 = (0 > (((c0 + -14) * 2 < 0?-(-(c0 + -14) / 2) : ((2 < 0?
          (-(c0 + -14) + - 2 - 1) / - 2 : (c0 + -14 + 2 - 1) / 2))))?
            0 : (((c0 + -14) * 2 < 0?-(-(c0 + -14) / 2) :
                ((2 < 0?(-(c0 + -14) + - 2 - 1) / - 2 : (c0 + -14 + 2 - 1) / 2)))));
         c2 <= ((1 < (((c0 + -2) * 2 < 0?
            ((2 < 0?-((-(c0 + -2) + 2 + 1) / 2) : -((-(c0 + -2) + 2 - 1) / 2))) :
            (c0 + -2) / 2))?1 : (((c0 + -2) * 2 < 0?
            ((2 < 0?-((-(c0 + -2) + 2 + 1) / 2) : -((-(c0 + -2) + 2 - 1) / 2))) : (c0 + -2) / 2))));
         c2++) {
        B[c0 + -2 * c2] = 0.33333 * (A[c0 + -2 * c2 - 1] + A[c0 + -2 * c2] + A[c0 + -2 * c2 + 1]);
        A[c0 + -2 * c2 + -1] = B[c0 + -2 * c2 + -1];
    }
    if (c0 <= 3) {
      if ((c0 + 1) % 2 == 0) {
        B[1] = 0.33333 * (A[1 - 1] + A[1] + A[1 + 1]);
      }
    }
  }
}
#pragma endscop
```

Fig. 2. Optimized Jacobi-1d-Imper benchmark (“tile_8_1_1” variant)

The state transition graph represents all possible states of a program. We use a symbolic representation of states where values and relations between variables are represented as predicates. If the concrete value of a variable is known, then all arithmetic operations are performed on this variable (without approximation). If no value is known (for example an input variable) then predicates are established to represent the set of possible states and a path-sensitive analysis is performed. The computation of the state transition graph (STG) has also been used in the RERS Challenge [17] 2012 and 2013 where the STG was then used for the verification of linear temporal logic formulas. For the verification of the polyhedral optimizations we use the STG to reason on the states of the program and extract the sequence of all floating point and array update operations that can be performed by the program. The implementation is integrated in our ROSE tool CodeThorn.

3.3 Floating Point Operation and Array Access Extraction

The floating point operation and array access extraction follows the reachability in the state transition graph (STG) from a selected node. In our case the selected node is the entry node of the function that contains the PolyOpt/C generated loop nest. These are marked in the PolyBench/C programs with pragmas (see Fig. 1 and 2).

Since we consider only the limited form of loops with constant numeric bounds, the entire state space of the loop can be computed in a form that is equivalent to loop unrolling. Our analyzer can also extract predicates from conditions for variables with unknown values. For the benchmarks this is not relevant though, because variables in those conditions have no data dependence on input values. The benchmarks contain conditions inside the loop body, guarding array updates, but those conditions only contain loop iteration variables. Since the values for the loop iteration variables are determined by the analysis (when computing the state transition graph), those conditions can be evaluated as well. Therefore, for the polyhedral benchmarks, our path sensitive analyzer can establish exactly one execution path for each given benchmark. From this determined state transition sequence we extract the terms of the floating point variable updates (including arrays). Only those terms representing variable updates and its corresponding state (containing a property state with a mapping of each iteration variable to a value) are relevant for the remaining phases.

3.4 Rewrite Rules

We establish a small set of rewrite rules which are sufficient to verify the given benchmarks. The rewrite rules operate on the extracted terms of the program representing floating point operations and array updates (as described in Section 3.3).

1. $Minus(IntVal.val) \Rightarrow IntVal.val' = -IntVal.val$
2. $AddAssign(\$L, \$R) \Rightarrow Assign(\$L, Add(\$L, \$R))$
3. $SubAssign(\$L, \$R) \Rightarrow Assign(\$L, Sub(\$L, \$R))$
4. $MulAssign(\$L, \$R) \Rightarrow Assign(\$L, Mul(\$L, \$R))$
5. $DivAssign(\$L, \$R) \Rightarrow Assign(\$L, Div(\$L, \$R))$
6. $Add(IntVal_1, IntVal_2) \Rightarrow IntVal.val = IntVal_1.val\ +\ IntVal_2.val$
7. $Sub(IntVal_1, IntVal_2) \Rightarrow IntVal.val = IntVal_1.val\ -\ IntVal_2.val$
8. $Mul(IntVal_1, IntVal_2) \Rightarrow IntVal.val = IntVal_1.val\ *\ IntVal_2.val$
9. $Div(IntVal_1, IntVal_2) \Rightarrow IntVal.val = IntVal_1.val\ /\ IntVal_2.val$
10. If a variable v in term t has a constant value c in the associated state S in the STG, then replace the variable v with the constant c in the term t.

The rewrite rules are applied to each extracted assignment separately (i.e. each array assignment and floating point variable assignment). Rule 1 eliminates the unary minus operator. Rules 2-5 eliminate compound assignment operators that modify the same variable and replace it with the assignment operator. This step is a normalization step for the next phase where SSA Form is established. Keeping in mind that we want to verify the property of the polyhedra generated code that ensures that the lexicographic order is preserved, we do not want to reorder array updates. More specifically, we do not want to reorder any floating point operations. Rules 6-9 perform constant folding for terms with one arithmetic operator and two constant operands. In Rule 10 the variables are replaced with the constant value that has been established in the state transition graph for that variable. Note that this rule is crucial as the iteration variables in the original program and generated variants have different names. Since this rule eliminates variable names, matching can be performed based on array variable index values.

The rewrite rules are applied on each assignment representing an array update or floating point operation (which may contain array access expressions with index computations) until no rule can be applied anymore. Expressions that are not matched by the rewrite rules remain unchanged. The rewrite rules guarantee termination by design.

3.5 Verification

The verification steps consist of representing all indexed array elements as unique variables (each array element is considered as one unique variable), generating Static Single Assignment Form [16] for the sequence of array updates and floating point operations, and the final equality test (matching) of the SSA forms.

Represent each Variable and Array Element as One Variable and Generate SSA Form. In Fig. 3 the final result for our running example Jacobi-1d is shown. The expressions of all assignments to arrays have been rewritten applying rules 1-10 (see Section 3.4). Each array element is treated as one separate variable. For example, `a[0]` is treated as a different variable to `a[1]`. The sequence of array operations (of the entire program extracted from the STG) is shown. For this sequence we establish SSA Form. The SSA numbers are postfixed to each element. For example `B[1]=...`; `A[1]=B[1]` becomes `B[1]_1=...`; `A[1]_1=B[1]_1`. Note that the array notation is only for readability. At this stage it is only relevant to have a unique name for each memory cell (i.e. we could also rename the array element to `B_1_1`)

This step is similar with the compiler optimization of scalar replacement of aggregates (SRoA) and variables as also performed by LLVM after loop unrolling. In particular, LLVM also generates an SSA Form after this replacement. Thus, our approach is in this respect similar to existing analyses and transformations in existing compilers and may be suitable for a verifying compiler that also checks whether the transformed program preserves the program semantics w.r.t. the sequence of floating point operations. Note that SRoA is usually only applied up to a certain size of an aggregate as well.

The SSA numbering allows to define a set of assignments while preserving all data dependencies. If the sets are equal for two given programs, they are guaranteed to have the same sequence of updates and operations on all floating point variables independent from their values. Note that the benchmarks do contain conditionals inside the loop. But those can be completely resolved in the computation of the state transition graph when applied to integers for which constant values can be inferred. In cases where the value is unknown (e.g. a test on floating point values) the term remains in the expression; i.e. the analyzer performs a partial evaluation of the program and the non-evaluated part is represented as term. For the given benchmarks the SSA Form for the *extracted* sequence of updates does not require phi-assignments. The reason is that the benchmarks only contain conditionals on index variables which become constant after unrolling. Also see Fig. 2 for an example of a benchmark code with such properties. The ternary operator inside expressions is used inside floating point

Jacobi-1D-Imper (original)	Jacobi-1D-Imper (Variant tile_8_1_1)
B[1]_1 = 0.33333 *(A[0]_0 + A[1]_0 + A[2]_0)	B[1]_1 = 0.33333 *(A[0]_0 + A[1]_0 + A[2]_0)
B[2]_1 = 0.33333 *(A[1]_0 + A[2]_0 + A[3]_0)	B[2]_1 = 0.33333 *(A[1]_0 + A[2]_0 + A[3]_0)
B[3]_1 = 0.33333 *(A[2]_0 + A[3]_0 + A[4]_0)	A[1]_1 = B[1]_1
B[4]_1 = 0.33333 *(A[3]_0 + A[4]_0 + A[5]_0)	B[3]_1 = 0.33333 *(A[2]_0 + A[3]_0 + A[4]_0)
B[5]_1 = 0.33333 *(A[4]_0 + A[5]_0 + A[6]_0)	A[2]_1 = B[2]_1
B[6]_1 = 0.33333 *(A[5]_0 + A[6]_0 + A[7]_0)	B[1]_2 = 0.33333 *(A[0]_0 + A[1]_1 + A[2]_1)
B[7]_1 = 0.33333 *(A[6]_0 + A[7]_0 + A[8]_0)	B[4]_1 = 0.33333 *(A[3]_0 + A[4]_0 + A[5]_0)
B[8]_1 = 0.33333 *(A[7]_0 + A[8]_0 + A[9]_0)	A[3]_1 = B[3]_1
B[9]_1 = 0.33333 *(A[8]_0 + A[9]_0 + A[10]_0)	B[2]_2 = 0.33333 *(A[1]_1 + A[2]_1 + A[3]_1)
B[10]_1 = 0.33333 *(A[9]_0 + A[10]_0 + A[11]_0)	A[1]_2 = B[1]_2
B[11]_1 = 0.33333 *(A[10]_0 + A[11]_0 + A[12]_0)	B[5]_1 = 0.33333 *(A[4]_0 + A[5]_0 + A[6]_0)
B[12]_1 = 0.33333 *(A[11]_0 + A[12]_0 + A[13]_0)	A[4]_1 = B[4]_1
B[13]_1 = 0.33333 *(A[12]_0 + A[13]_0 + A[14]_0)	B[3]_2 = 0.33333 *(A[2]_1 + A[3]_1 + A[4]_1)
B[14]_1 = 0.33333 *(A[13]_0 + A[14]_0 + A[15]_0)	A[2]_2 = B[2]_2
A[1]_1 = B[1]_1	B[6]_1 = 0.33333 *(A[5]_0 + A[6]_0 + A[7]_0)
A[2]_1 = B[2]_1	A[5]_1 = B[5]_1
A[3]_1 = B[3]_1	B[4]_2 = 0.33333 *(A[3]_1 + A[4]_1 + A[5]_1)
A[4]_1 = B[4]_1	A[3]_2 = B[3]_2
A[5]_1 = B[5]_1	B[7]_1 = 0.33333 *(A[6]_0 + A[7]_0 + A[8]_0)
A[6]_1 = B[6]_1	A[6]_1 = B[6]_1
A[7]_1 = B[7]_1	B[5]_2 = 0.33333 *(A[4]_1 + A[5]_1 + A[6]_1)
A[8]_1 = B[8]_1	A[4]_2 = B[4]_2
A[9]_1 = B[9]_1	B[8]_1 = 0.33333 *(A[7]_0 + A[8]_0 + A[9]_0)
A[10]_1 = B[10]_1	A[7]_1 = B[7]_1
A[11]_1 = B[11]_1	B[6]_2 = 0.33333 *(A[5]_1 + A[6]_1 + A[7]_1)
A[12]_1 = B[12]_1	A[5]_2 = B[5]_2
A[13]_1 = B[13]_1	B[9]_1 = 0.33333 *(A[8]_0 + A[9]_0 + A[10]_0)
A[14]_1 = B[14]_1	A[8]_1 = B[8]_1
B[1]_2 = 0.33333 *(A[0]_0 + A[1]_1 + A[2]_1)	B[7]_2 = 0.33333 *(A[6]_1 + A[7]_1 + A[8]_1)
B[2]_2 = 0.33333 *(A[1]_1 + A[2]_1 + A[3]_1)	A[6]_2 = B[6]_2
B[3]_2 = 0.33333 *(A[2]_1 + A[3]_1 + A[4]_1)	B[10]_1 = 0.33333 *(A[9]_0 + A[10]_0 + A[11]_0)
B[4]_2 = 0.33333 *(A[3]_1 + A[4]_1 + A[5]_1)	A[9]_1 = B[9]_1
B[5]_2 = 0.33333 *(A[4]_1 + A[5]_1 + A[6]_1)	B[8]_2 = 0.33333 *(A[7]_1 + A[8]_1 + A[9]_1)
B[6]_2 = 0.33333 *(A[5]_1 + A[6]_1 + A[7]_1)	A[7]_2 = B[7]_2
B[7]_2 = 0.33333 *(A[6]_1 + A[7]_1 + A[8]_1)	B[11]_1 = 0.33333 *(A[10]_0 + A[11]_0 + A[12]_0)
B[8]_2 = 0.33333 *(A[7]_1 + A[8]_1 + A[9]_1)	A[10]_1 = B[10]_1
B[9]_2 = 0.33333 *(A[8]_1 + A[9]_1 + A[10]_1)	B[9]_2 = 0.33333 *(A[8]_1 + A[9]_1 + A[10]_1)
B[10]_2 = 0.33333 *(A[9]_1 + A[10]_1 + A[11]_1)	A[8]_2 = B[8]_2
B[11]_2 = 0.33333 *(A[10]_1 + A[11]_1 + A[12]_1)	B[12]_1 = 0.33333 *(A[11]_0 + A[12]_0 + A[13]_0)
B[12]_2 = 0.33333 *(A[11]_1 + A[12]_1 + A[13]_1)	A[11]_1 = B[11]_1
B[13]_2 = 0.33333 *(A[12]_1 + A[13]_1 + A[14]_1)	B[10]_2 = 0.33333 *(A[9]_1 + A[10]_1 + A[11]_1)
B[14]_2 = 0.33333 *(A[13]_1 + A[14]_1 + A[15]_0)	A[9]_2 = B[9]_2
A[1]_2 = B[1]_2	B[13]_1 = 0.33333 *(A[12]_0 + A[13]_0 + A[14]_0)
A[2]_2 = B[2]_2	A[12]_1 = B[12]_1
A[3]_2 = B[3]_2	B[11]_2 = 0.33333 *(A[10]_1 + A[11]_1 + A[12]_1)
A[4]_2 = B[4]_2	A[10]_2 = B[10]_2
A[5]_2 = B[5]_2	B[14]_1 = 0.33333 *(A[13]_0 + A[14]_0 + A[15]_0)
A[6]_2 = B[6]_2	A[13]_1 = B[13]_1
A[7]_2 = B[7]_2	B[12]_2 = 0.33333 *(A[11]_1 + A[12]_1 + A[13]_1)
A[8]_2 = B[8]_2	A[11]_2 = B[11]_2
A[9]_2 = B[9]_2	A[14]_1 = B[14]_1
A[10]_2 = B[10]_2	B[13]_2 = 0.33333 *(A[12]_1 + A[13]_1 + A[14]_1)
A[11]_2 = B[11]_2	A[12]_2 = B[12]_2
A[12]_2 = B[12]_2	B[14]_2 = 0.33333 *(A[13]_1 + A[14]_1 + A[15]_0)
A[13]_2 = B[13]_2	A[13]_2 = B[13]_2
A[14]_2 = B[14]_2	A[14]_2 = B[14]_2

Fig. 3. Example: Extracted assignments (updates) from the programs in Fig. 1 (left column) and Fig. 2 (right column) after rewrite and renaming in SSA Form. The rewrite rules that are applied are those listed in Section 3.4. Rewrite statistics for this example are shown in Table 2 (see rows for jacobi-1d-imper and jacobi-1d-imper-tile-8-1-1). The number of extracted assignments is 56 and the number of applied rewrite operations differs significantly, but the final results are two sequences of assignments that are equivalent - they do differ in the interleaving of the assignments, but the order of updates on all SSA variables is identical. This is checked by Algorithm 2 after sorting both sequences by the updated SSA variable.

```
stddev[0]_18 = stddev[0]_17 / float_n_0
stddev[0]_19 = sqrt(stddev[0]_18)
stddev[0]_20 =(stddev[0]_19 <= eps_0?1.0 : stddev[0]_19)
stddev[1]_1 = 0.0
stddev[1]_2 = stddev[1]_1 +(data[0][1]_0 - mean[1]_18) *(data[0][1]_0 - mean[1]_18)
```

Fig. 4. Fragment of verified update sequence for datamining/correlation benchmark

computations though. But since we represent in our analysis all floating point values to be an unknown value, the different states that are established by the analyzer during the expression analysis are determined to be equal, and thus, only a single state transition is established in the STG for such an assignment (see Fig. 4) and the extracted computation remains a sequence of assignments for the given programs. For each PolyBench/C 3.2 benchmark exactly one sequence of computations can be extracted.

Equality Test of SSA Forms. The final step in the verification is to determine the equivalence of the SSA Forms of two program variants. Our approach considers a verification to be successful if the term representations of the set of assignments in SSA Form is equal.

The interleaving of the assignments may differ as is demonstrated also in the example in Fig. 3, but the sequence of updates for each array element must be exactly the same. In particular, also the operations on the rhs of each assignment must match exactly (term equivalence). For the example in Fig. 3 this is indeed the case. For example the sequence of updates on the elements of `B[1]_1`, `B[2]_1` `B[3]_1` etc. is exactly the same in both columns. This holds for all SSA enumerated variables.

For all the evaluated benchmarks the extracted SSA Forms match exactly as unordered sets. The small set of rewrite rules presented in Section 3.4 is sufficient to proof equivalence for the Polybech/C 3.2 benchmarks.

Supported Language Subset. In Fig. 4 a fragment of the update sequence for the correlation benchmark is shown. It includes the use of a floating point variable which is assigned a value outside the polyhedral optimized program section and therefore has SSA number 0, the ternary operator, an external function call (sqrt), and a computation involving different arrays and their elements. *This code does not have a static control-flow*, because of the ternary operator leading to a data-dependent assignement of the value. Previous work on affine program equivalence [8] cannot handle such case, in contrast our approach supports such construct.

For a defined set of external function calls we assume that the functions are side effect free (e.g. sqrt). For each PolyBench/C benchmark and PolyOpt/C generated variant with constant array bounds, our analysis can determine an STG with exactly one floating point computation sequence. We consider cases where more than one execution path is represented in the STG in our future work.

Error Detection. When the equality test fails, then the semantic equivalence of two programs cannot be established. This can have two reasons i) the programs are different, ii) our rewrite system is not powerful enough to establish a normalized representation such that the two programs' sequence of floating-point operations can be matched. For two benchmarks we determined differences in the update sequence (cholesky and reg_detect), as shown in Sec. 4.3. The difference is reported as the set of non-matching assignments.

4 Results

Our implementation is based on ROSE [18]. The computation of the state transition graph (STG) has also been used in the RERS Challenge [17] 2012 and 2013.

The rewrite system is based on the AstMatching mechanism in ROSE and implements the small number of rules that turned out to be sufficient to verify benchmarks of the PolyBench/C suite.

Benchmarks in PolyBench/C 3.2 contain SCoPs and represent computation in linear algebra, datamining, and stencil computing. PolyOpt/C performs data dependence analysis, loop transformation and code generation based on the polyhedral model. Because of the transformation capability and its integration in the ROSE compiler, PolyOpt/C is chosen as the optimization driver in the experiments. Optimization variants in this study are mainly generated from the following two transformations:

– Tiling-driven transformations: the "–polyopt-fixed-tiling" option in PolyOpt/C implements arbitrarily complex sequences of loop transformations to maximize the tilability of the program, applies tiling, and expose coarse-grain or wavefront parallelism between tiles [2]. It allows to specify the tile size to be used in each tiled dimension.
– Data locality-driven transformations: we use the Pluto algorithm [2] for maximal data locality, and test the three different statement fusion schemes (minfuse, maxfuse and smartfuse) implemented in PolyOpt/C.

Figure 5 illustrates the transformation flow. Each benchmark code with a constant array size (size in each dimension) is given to PolyOpt/C. The following variants are generated for the study:

1. Loop fusion: Polyhedral transformation performs multiple transformations in a single phase. In this variant, we apply the three fusion schemes: maxfuse, smartfuse and nofuse to drive the loop transformation. A combination of loop permutation, skewing, shifting, fusion and distribution is implemented for each of the three statement fusion schemes.
2. Loop tiling: PolyOpt/C takes 3 parameters to form a tile size for single and multi-dimensional tiling. Tile sizes with number of power of two are commonly seen in real applications. Other special tile sizes, such as size in prime numbers, or non-tilable sizes (size larger than problem/array size), are

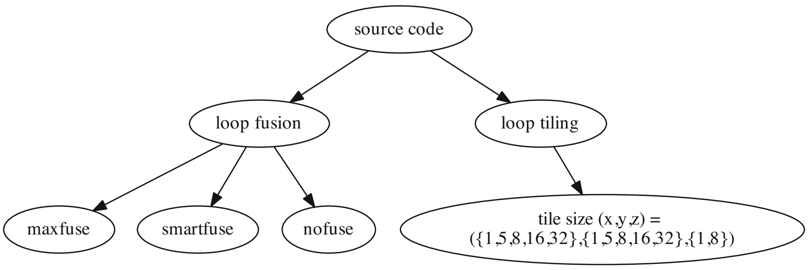


Fig. 5. Transformation variants: 53 variants are generated for each benchmark (3 for fusion and $5 \times 5 \times 2 = 50$ for tiling)

also included to prove a broader span for verification: the polyhedral code generator CLooG [11] integrated in PolyOpt/C generates a possibly different code for each of these cases. Some tile sizes might not be applicable to all benchmarks (e.g., those which cannot be tiled along multiple dimensions), but PolyOpt/C still generates a valid output with 0D or 1D tiling for the verification. Note that we also want to verify certain corner cases in the code generation.

With our approach we can verify all PolyOpt/C generated optimization variants for PolyBench/C 3.2. As shown in Table 1) PolyOpt/C generated 53 variants for all but two benchmarks. Unfortunately, for 'doitgen' only the 3 fusion variants could be generated, and for dynprog the fusion and tiling variants could not be generated (due to an error in PolyOpt/C 0.2). For all other benchmarks all variants were generated (in total 53×28). We verified $53 \times 28 + 3 = 1487$ variants, and found errors in the fusion and the tiling variants of the cholesky benchmark and errors in the tiling variants of 'reg_detect' (in total $3 + 50 + 50 = 103$). The array size is set to 16 for each dimension in the 1D ,2D, and 3D cases and stepsize is set to 2 (stepsize is the time dimension for stencil benchmarks). The total verification time for all 1487 variants is 1.5 hours, as shown in column 3 in Table 1.

We recall a critical aspect of affine programs is that they have a control flow that only depends on loop iterators and program parameters. That is, while only one C program is generated by PolyOpt/C, it is by construction expected to be valid for any value the program parameters can take (e.g., the array size `N`). Checking the correctness of this code for a small array size (e.g., `N` = 16) still stresses the entire code for the benchmarks we considered.

4.1 State Space and Rewrite Operations Statistics

In Table 2 detailed statistics on some of the benchmarks are shown. The statistics include the number of computed states in the state transition graph and how often each rewrite rule has been applied for a benchmark variant. The original program is listed by the name of the benchmark itself. Optimization variants are denoted by the benchmark name prefixed with the variant name.

Table 1. The benchmarks in the Polybench/C 3.2 suite with information whether we successfully verified the PolyOpt/C generated variants. For two benchmarks our verification procedure helped to find a bug in some fusion and tiling optimizations in PolyOpt/C 0.2.

Benchmark	Verification (Size 16)	Total Run Time
2mm, 3mm, adi, atax, bicg, covariance, durbin, fdtd-2d, gemm, gemver, gesummv, jacobi-1d, jacobi-2d, lu, ludcmpm, mvt, seidel, symm, syr2k, syrk, trisolv, trmm, correlation, gramschmidt, fdtf-ampl, floyd-warshall	for 26 benchmarks all 53 variants verified	1h:30m:02s
dotitgen	all fusion variants verified	
cholesky	errors found in fusion and tiling opt	
reg_detect	errors found in tiling opt	
doitgen	tiling variants not available	-
dynprog	variants not available	-

4.2 Run Times

The run times for each benchmark and each generated optimization variant is shown in the last column in Table 2. This includes all phases (parsing, STG analysis, update extraction, update normalization, and sorting of the update sequence). The total verification time of an original program and a variant is the sum of the total time (as shown in Table 2 of both entries in the table plus the time for comparison. The final comparison is linear in the number of assignments because Algorithm 1 also includes sorting by the unique SSA assignment variable on lhs of each update operation.

For example, to verify that the benchmark `jacobi-1d-imper` and the polyhedral optimization variant `tile_8_1_1` are equivalent, the total verification time is the sum of the run times for the original benchmark and the variant (each one shown in last column in Table 2) and the time for matching the assignments of the two sorted lists of assignments (not shown). Note that this is also valid for any pair of variants for the same benchmark, hence the original benchmark only needs to be analyzed once. The total run time for the verification of all 1487 generated benchmark variants, including all operations, is shown in Table 1.

4.3 Bug Found Thanks to Verification

While our verification asserted the correctness of 1383 different program variants generated by PolyOpt/C for the tested problem sizes, a significant outcome is the finding of a previously unknown bug, resulting in differences in cholesky and reg_detect variants. For cholesky we determined that one assignment pair (for `A[1][0]_1`) does not match (see Fig. 6). All other 951 assignments do match.

The current test suite of PolyOpt/C checks the correctness of the transformed code by checking if the output of the computation is strictly identical for the reference and transformed codes. Under this scheme, errors that amount to changing the order of two update operations (thereby violating the dependence between such operations) may not be caught: in practice, IEEE floating point operations are often commutative/associative and therefore changing the order of their com-

Table 2. Shows from left to right for some selected benchmark results: the number of computed states in the STG, number of extracted assignments (updates), how often the rewrite rules 1-10 were applied, and the run time

Benchmark-Variant	States	Updates	R1	R2-5	R6-9	R10	Run Time
3mm	40922	13056	0	12288	0	75264	5.72 secs
3mm-fuse-smartfuse	40925	13056	0	12288	0	75264	5.74 secs
3mm-tile-8-1-1	50916	13056	0	12288	0	75264	6.18 secs
covariance	9125	2992	0	2704	0	15440	1.50 secs
covariance-fuse-smartfuse	9128	2992	0	2704	0	15440	1.50 secs
covariance-tile-8-1-1	12652	2992	0	2704	0	15440	1.62 secs
fdtd-2d	4731	1442	0	0	1860	13144	1.49 secs
fdtd-2d-fuse-smartfuse	4734	1442	0	0	1860	13144	1.52 secs
fdtd-2d-tile-8-1-1	5436	1442	17700	0	28260	21356	2.94 secs
fdtd-apml	52829	34816	0	0	12544	353792	23.33 secs
fdtd-apml-fuse-smartfuse	52832	34816	0	0	12544	353792	23.22 secs
fdtd-apml-tile-8-1-1	60171	34816	0	0	12544	353792	23.66 secs
floyd-warshall	13388	4096	0	0	0	57344	2.98 secs
floyd-warshall-fuse-smartfuse	13391	4096	0	0	0	57344	3.00 secs
floyd-warshall-tile-8-1-1	15776	4096	0	0	0	57344	3.10 secs
gramschmidt	14072	4504	0	2176	0	29712	2.11 secs
gramschmidt-fuse-smartfuse	14120	4504	0	2176	0	29712	2.14 secs
gramschmidt-tile-8-1-1	18924	4504	0	2176	0	29712	2.31 secs
jacobi-1d-imper	194	56	0	0	56	168	371.06 ms
jacobi-1d-imper-fuse-smartfuse	196	56	0	0	56	168	369.01 ms
jacobi-1d-imper-tile-8-1-1	232	56	208	0	420	312	396.92 ms
jacobi-2d-imper	2602	784	0	0	1568	6272	663.44 ms
jacobi-2d-imper-fuse-smartfuse	2605	784	0	0	1568	6272	663.97 ms
jacobi-2d-imper-tile-8-1-1	3923	784	7192	0	15032	11680	1.46 secs
seidel-2d	1309	392	0	0	4704	7840	1.03 secs
seidel-2d-fuse-smartfuse	1312	392	0	0	4704	7840	1.02 secs
seidel-2d-tile-8-1-1	2144	392	11760	0	28224	19600	2.28 secs
trmm	6794	1920	0	1920	0	11520	1.22 secs
trmm-fuse-smartfuse	6797	1920	0	1920	0	11520	1.24 secs
trmm-tile-8-1-1	9438	1920	3840	1920	7680	15360	2.39 secs

Benchmark	Original Program	Variant tile_8_1_1 (Detected Errors)
cholesky	`A[1][0]_1 = x_2 * p[0]_1`	`A[1][0]_1 = x_1 * p[0]_1`
reg_detect	`mean[0][0]_1 = sum_diff[0][0][15]_1`	`mean[0][0]_1 = sum_diff[0][0][15]_0`
	`mean[0][0]_2 = sum_diff[0][0][15]_2`	`mean[0][0]_2 = sum_diff[0][0][15]_1`
	`mean[0][1]_1 = sum_diff[0][1][15]_1`	`mean[0][1]_1 = sum_diff[0][1][15]_0`
	`mean[0][1]_2 = sum_diff[0][1][15]_2`	`mean[0][1]_2 = sum_diff[0][1][15]_1`
	`mean[1][1]_1 = sum_diff[1][1][15]_1`	`mean[1][1]_1 = sum_diff[1][1][15]_0`
	`mean[1][1]_2 = sum_diff[1][1][15]_2`	`mean[1][1]_2 = sum_diff[1][1][15]_1`

Fig. 6. Errors found in generated optimized programs. The equality check in Algorithm 2 reported semantic inequality because the right-hand-sides of some corresponding assignments are different. Shown are those floating-point operations that do not match.

putation may not always lead to a different final result. The clear merit of our approach is demonstrated by finding a bug that could hardly be caught by classical testing means, but was immediately found by our verification process. In addition, the ability to point to the set of operations that do not match greatly helps the bug finding process.

5 Related Work

Existing research adopts various approaches to verify the transformation results. Focusing on affine programs, Verdoolaege et al. develop an automatic equivalence proofing [8]. The equivalence checking is heavily dependent upon the fact that input programs have an affine control-flow, as the method is based on mathematical reasoning about integer sets and maps built from affine expressions, and the development of widening/narrowing operators to properly handle non-uniform recurrences. In contrast, our work has a strong potential for generalization beyond affine programs. In fact, we already support some cases of data-dependent control-flow in the verification, something not supported by previous work [8].

Karfa et al. also designed a method exclusively for a subset of affine programs, using array data dependence graphs (ADDGs) to represent the input and transforming behaviors. An operator-level equivalence checking provides the capability to normalize the expression and establish matching relations under algebraic transformations [19]. Mansky and Gunter [20] use the TRANS language [21] to represent transformations. The correctness proof is verified by Isabelle [22], a generic proof assistant, implemented in the verification framework.

6 Conclusion

We have presented an approach for verifying that the implementation of PolyOpt/C for polyhedral optimizations is semantics preserving. Our approach first performs a static analysis and determines a list of terms representing the updates on floating point variables and array elements. This sequence is then rewritten by a rewrite system and eventually an SSA Form is established where each array element is treated as a separate variable. When the sets of array updates are equal and all terms match exactly then we have determined that the programs are indeed semantically equivalent. Otherwise we do not know whether the programs are equivalent or not. With our approach we were able to verify all PolyOpt/C 0.2 generated variants for PolyBench/C 3.2, out of which 1384 variants were shown to be correct, and we found errors in 103 generated variants, corresponding to one bug occuring for two benchmarks. This bug was not previously known and was not caught by the existing test suite of PolyOpt/C, which is based only on checking that the output data produced by the transformed code is identical to the output produced by the reference code. We limited our evaluation to a size of 16 (for each array dimension) because our approach requires to analyze the entire state space of the loop iterations and we wanted to keep the overall verification time for all benchmarks and variants within a few hours, such that the verification procedure can be used in the release process of PolyOpt/C in future.

References

1. Pouchet, L.N.: PolyOpt/C 0.2.0: A Polyhedral Compiler for ROSE (2012), `http://www.cs.ucla.edu/~pouchet/software/polyopt/`

2. Bondhugula, U., Hartono, A., Ramanujam, J., Sadayappan, P.: A practical automatic polyhedral program optimization system. In: ACM SIGPLAN Conference on Programming Language Design and Implementation (June 2008)
3. Kong, M., Veras, R., Stock, K., Franchetti, F., Pouchet, L.N., Sadayappan, P.: When polyhedral transformations meet simd code generation. In: PLDI (June 2013)
4. Holewinski, J., Pouchet, L.N., Sadayappan, P.: High-performance code generation for stencil computations on gpu architectures. In: ICS (June 2012)
5. Pouchet, L.N., Zhang, P., Sadayappan, P., Cong, J.: Polyhedral-based data reuse optimization for configurable computing. In: FPGA (February 2013)
6. Pouchet, L.N.: PoCC 1.2: The Polyhedral Compiler Collection (2012), `http://www.cs.ucla.edu/~pouchet/software/pocc/`
7. Leroy, X.: The Compcert C compiler (2014), `http://compcert.inria.fr/compcert-C.html`
8. Verdoolaege, S., Janssens, G., Bruynooghe, M.: Equivalence checking of static affine programs using widening to handle recurrences. ACM Transactions on Programming Languages and Systems (TOPLAS) 34(3), 11 (2012)
9. Feautrier, P.: Some efficient solutions to the affine scheduling problem, part II: Multidimensional time. Intl. J. of Parallel Programming 21(6), 389–420 (1992)
10. Irigoin, F., Triolet, R.: Supernode partitioning. In: ACM SIGPLAN Principles of Programming Languages, pp. 319–329 (1988)
11. Bastoul, C.: Code generation in the polyhedral model is easier than you think. In: IEEE Intl. Conf. on Parallel Architectures and Compilation Techniques (PACT 2004), Juan-les-Pins, France, pp. 7–16 (September 2004)
12. Pouchet, L.N., Bondhugula, U., Bastoul, C., Cohen, A., Ramanujam, J., Sadayappan, P., Vasilache, N.: Loop transformations: Convexity, pruning and optimization. In: POPL, pp. 549–562 (January 2011)
13. Girbal, S., Vasilache, N., Bastoul, C., Cohen, A., Parello, D., Sigler, M., Temam, O.: Semi-automatic composition of loop transformations. Intl. J. of Parallel Programming 34(3), 261–317 (2006)
14. Pouchet, L.N.: PolyBench/C 3.2 (2012), `http://www.cs.ucla.edu/~pouchet/software/polybench/`
15. Allen, J., Kennedy, K.: Optimizing Compilers for Modern Architectures. Morgan Kaufmann Publishers (2002)
16. Cytron, R., Ferrante, J., Rosen, B.K., Wegman, M.N., Zadeck, F.K.: Efficiently computing static single assignment form and the control dependence graph. ACM Transactions on Programming Languages and Systems 13(4), 451–490 (1991)
17. Steffen (Organizer), B.: RERS Challenge: Rigorous Examination of Reactive Systems (2010, 2012, 2013, 2014), `http://www.rers-challenge.org`
18. Quinlan, D., Liao, C., Matzke, R., Schordan, M., Panas, T., Vuduc, R., Yi, Q.: ROSE Web Page (2014), `http://www.rosecompiler.org`
19. Karfa, C., Banerjee, K., Sarkar, D., Mandal, C.: Verification of loop and arithmetic transformations of array-intensive behaviors. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 32(11), 1787–1800 (2013)
20. Klein, G.: A framework for formal verification of compiler optimizations. In: Kaufmann, M., Paulson, L. (eds.) ITP 2010. LNCS, vol. 6172, pp. 371–386. Springer, Heidelberg (2010)
21. Kalvala, S., Warburton, R., Lacey, D.: Program transformations using temporal logic side conditions. ACM Transactions on Programming Languages and Systems (TOPLAS) 31(4), 14 (2009)
22. Paulson, L.C.: Isabelle Page, `https://www.cl.cam.ac.uk/research/hvg/Isabelle`

The Guided System Development Framework: Modeling and Verifying Communication Systems

Jose Quaresma, Christian W. Probst, and Flemming Nielson

Technical University of Denmark
{jncq,cwpr,fnie}@dtu.dk

Abstract. In a world that increasingly relies on the Internet to function, application developers rely on the implementations of protocols to guarantee the security of data transferred. Whether a chosen protocol gives the required guarantees, and whether the implementation does the same, is usually unclear. The Guided System Development framework contributes to more secure communication systems by aiding the development of such systems. The framework features a simple modelling language, step-wise refinement from models to implementation, interfaces to security verification tools, and code generation from the verified specification. The refinement process carries thus security properties from the model to the implementation. Our approach also supports verification of systems previously developed and deployed. Internally, the reasoning in our framework is based on the Beliefs and Knowledge tool, a verification tool based on belief logics and explicit attacker knowledge.

1 Introduction

Developing secure communication systems is difficult. Application developers rely on the implementations of protocols to guarantee the security of data transferred. Whether a chosen protocol gives the required guarantees, and whether the implementation does the same, is usually unclear.

Verifying secure communication systems is difficult, too, though for different reasons. While a plethora of formal approaches and tools for protocol verification exist, they are often not accessible to developers, and only connect the guarantees to the implementation of the protocol.

The Guided System Development (GSD) framework aims at making development and verification of secure communication systems easier by bridging the gap between system development and verification of communication protocols. The knowledge and skills required to successfully use a security verification tool are significant. With the GSD framework [1,2] it is possible to use such tools and have access to their results *without* the need for that specific knowledge.

The main achievement of the GSD framework is to make building secure communication systems the *only* option. This is reached through a number of components: a simple and intuitive modelling language, step-wise refinement of guarantees from the model to its implementation, built-in interfaces to established security verification tools, and finally code generation.

The rest of the paper is structured as follows: after an introduction of a running example we use throughout the paper, Sec. 2 discusses related work. In Sec. 3 we present an overview of the framework. We then introduce the Abstract Global level in Sec. 4 and the Concrete level in Sec. 5. In Sec. 6, we present the interface to verification tools and code output. We evaluate the tool in Sec. 7 by presenting its usage on the modelling and verification of the Automatic Dependent Surveillance-Broadcast (ADS-B) system, a new air traffic management surveillance system that is replacing the radar as the main means for traffic management in the near future. Finally, Sec. 8 concludes the paper and discusses future work.

1.1 The Message Board

To illustrate the use of GSD, we use the communication behind a message board as an example. This message board allows a user to share a message either anonymously and/or confidentially. The combination of these properties gives rise to four different kinds of messages:

1. The message sent can be seen by everybody without any guarantees regarding the identity of the sender,
2. The message sent can be seen by everybody and it has guarantees regarding the identity of the sender,
3. The message sent can only be seen by a particular user without guarantees regarding the identity of the sender, or
4. The message sent can only be seen by a particular user and it has guarantees regarding the identity of the sender

For space reasons, we only model sending a message in an authenticated way, *i.e.*, that other users have guarantees regarding the message's author.

2 Related Work

CaPiTo [3] connects the abstract specification of Service-Oriented Systems with the standard protocol suites used in industry in an independent way, *i.e.*, the description of the system in terms of messages exchanged is separated from the description of which standard suites are going to be used. This separation of concerns when modelling a Service-Oriented System greatly inspired our framework. Furthermore, CaPiTo also allows the verification of the modeled protocols and the code generation of parts of the system. This is accomplished by providing a specification language, means to verify the security of the specified protocol, and the translation from the protocol specification language into an executable language. Compared with CaPiTo, our modelling language is simpler and more intuitive, it has views of different levels of the system, and connects to more verification tools, including a GSD-specific tool introduced in Sec. 6.1.

AVANTSSAR aims at validating trust and security of Service-Oriented Architectures. Compared with the GSD framework, the AVANTSSAR project[1] requires a higher level of expertise to start using it and does not provide automatic code generation, functionality that we believe is important when providing tools to help system designers and programmers.

A similar tool to the GSD framework is the Automatic Generation, Verification and Implementation of Security Protocols (AGVI) toolkit [4], that allows the system designer to describe the security requirements and the system specification. In AGVI a *protocol generator* creates candidate protocols that satisfy the given system requirements. After that, the protocols are analysed [5] and the ones that do not satisfy the desired security properties are discarded. Finally, a code generator translates the formal protocol specification into Java code. In comparison to AGVI, the GSD modelling language focusses on properties of communication. Furthermore, the GSD frameworks can by design be extended new formal verification tools, new output languages, new abstract ways of representing message security properties in the modelling language (extending the security modules presented in Sec. 4.2), and also new ways of implementing the different security modules.

There also exist more targeted work on modeling and implementation of secure communication systems. When compared to the GSD framework, these tools usually feature more concrete and complex modelling languages and target a specific verification tool. FS2PV [6] derives a formal model from a protocol code written in F (a first-order subset of F#) and symbolic libraries. The translation is made to π-calculus, which can then be verified by ProVerif [7]. Swamy *et al.* [8] developed a dependently typed language (F*) aimed at secure distributed programming. Programs written in F* are translated to .NET bytecode. Other more recent work [9] enables the verification of a protocol with CryptoVerif [10] and then translates that specification into OCaml.

3 Framework Overview

The overall structure of the GSD framework is shown in Fig. 1. The framework is composed by three levels that represent different levels of abstraction. System developers specify the desired system at the *Abstract Global* level, using security modules (Sec. 4.2), to specify the required security assurances for the data being exchanged in the modeled system.

By unfolding the security modules using plugins, the Abstract Global level is transformed into a model of the system in the *Concrete Global* level. Plugins connect the abstract modules of the Abstract Global level with their implementations. For instance, if there is a security module in the Abstract Global specification that requires some data to be sent in a confidential way, this translation would replace it with an implementation of, for example, the Transport Layer Security (TLS) cryptographic protocol. The use of plugins, which is similar to the work by Gao *et al.* [3], separates the desired security assurances of the

[1] `http://www.avantssar.eu`, last accessed May 2014.

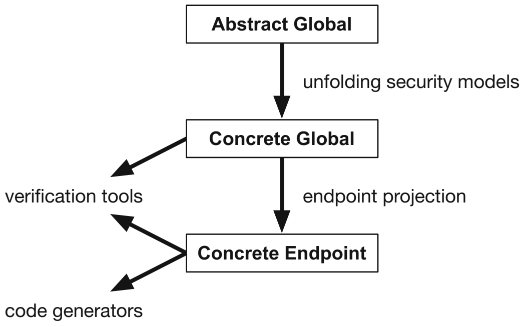


Fig. 1. Overview of the Guided System Development framework

exchanged data and the way to provide those assurances. In Sec. 5.1, we discuss the use of plugins and the flexibility that they provide.

A specification at the Concrete Global level is made more concrete using endpoint projection, resulting in the *Concrete Endpoint level.* This step separates the specifications for each individual, and is closer to the final implementation and to some of the verification tools (Sec. 5.2).

Contracts, another component of GSD, represent the desired outcomes of the different security modules in the Abstract Global level; they describe the modules' semantics. For example, part of the contract for the confidentiality module expresses that only the intended recipient of a message sent confidentially is able to read the message. Contracts enable some preliminary reasoning in the Abstract Global level, and the verification of the implementations of each security module by comparing the desired outcomes with the outcomes achieved by the different implementations.

3.1 Framework Inputs and Outputs

The modeling language used in GSD is strongly influenced by Alice and Bob notation, a simple and intuitive way of modelling communication system. The example in Fig. 2 uses this notation to model a message `msg` being sent by a principal called `User` to another principal called `Board`. The complete language syntax for the input language (in the Abstract Global level) is presented in Fig. 3.

When building a system with secure communications using GSD, the input for the framework is the specification of the system in the Abstract Global level, which includes security assurances for the exchanged messages. The language at this level is simple and intuitive and when using it, the step-wise refinement will lead to an implementation that provides the specified security assurances.

```
User → Board : msg
```

Fig. 2. Example of sending a message from `User` to `Board` in Alice and Bob notation

$$
\begin{aligned}
\text{system} &::= \text{stm;} \mid \text{system stm;}\\
\text{stm} &::= \text{principal} \rightarrow \text{principal} : \text{msgs}\\
\text{principal} &::= \textit{string}\\
\text{msgs} &::= \text{msg} \mid \text{msgs, msg}\\
\text{msg} &::= \text{el} \mid \text{secModule}\\
\text{el} &::= \textit{string}\\
\text{secModule} &::= \text{secAssurance(args)}\\
\text{secAssurance} &::= \mathit{Auth} \mid \mathit{freshAuth} \mid \mathit{Conf} \mid \mathit{Sec} \mid \mathit{freshSec}\\
\text{args} &::= \textit{string} \mid \text{args, } \textit{string}
\end{aligned}
$$

Fig. 3. Syntax of the language in the Abstract Global level

It is, however, also possible to use the framework by writing the specification in one of the other two abstraction levels presented before. If so, one would not take full benefit of GSD, but would still benefit from some of the connections to the verification tools and code outputs. We believe that this option is useful for more experienced system developers or for already implemented systems, in which case a specification closer to the implementation might be easier to write.

The outputs of GSD can be divided into two categories: the information from the supported verification tools and the implementation of the modeled system. We present these in more detail in Sec. 6.

4 Abstract Global Level

The goal at this level is to provide the developer with a simple and intuitive language that has the necessary tools to model the communication system that is being developed. As shown in Fig. 3, and as mentioned in Sec. 3.1, this language is similar to the Alice and Bob notation but extends that notation with security modules, which are presented in Sec. 4.2.

The logic used to express (and reason about) security properties is based on BAN logic [11] and more generally on SVO logic [12], a logic that unifies several different belief logics, including BAN logic itself. We use BAN and SVO to reason about the beliefs of the principals involved in the different message exchanges. Based on the beliefs at the end of a series of message exchanges, we are able to argue about security properties of the exchanged messages.

4.1 The Logic

BAN and SVO logics focus on the beliefs that legitimate principals are able to infer from a message exchange, and target authentication. Such logics are not less suited to directly reason about confidentiality, since confidentiality concerns what some non-legitimate principal might, or might not, be able to see from a

P Received el	- principal P received an Element el;
P Sees el	- principal P sees a specific Element el;
P Believes t	- principal P believes in a specific Term t;
P Said el	- principal P said, at some point in time, a specific Element el;
P Says el	- principal P recently said the Element el;
Conc(el1,el2)	- concatenation of two Elements;

Fig. 4. Terms in our logic

message exchange. There are several approaches to handle confidentiality in this case. We extend belief logics with explicit reasoning about the knowledge of non-legitimate principals: the beliefs they are able to infer from the message exchange and what they are able to see and (most importantly for the confidentiality reasoning) what they *not* are able to see about the exchanged messages. The chosen approach results in a simple model that is easy to reason about.

We consider non-legitimate principals to be Dolev-Yao attackers [13], *i.e.*, they are not only able to see all the exchanged messages but also capable of initiating protocol communications with legitimate principals and to forge new messages based on acquired knowledge.

Our logic is composed by principals and elements (all the artifacts that can be sent from one principal to another). The different terms in our logic are shown in Fig. 4. The rules used to reason about this logic are introduced in Sec. 6.1.

```
User → Board : Auth(User, message)
```

Fig. 5. Specification of `User` sending a message to the `Board` in an authenticated way

4.2 The Security Modules

The most important elements at the Abstract Global level are the security modules, which model security assurances for data exchanges:

- **None** is not actually a security module, but sends data in plaintext.
- **Auth** sends data such that the receiver can identify the sender.
- **Strong Auth** adds freshness to the Auth module to prevent replay attacks.
- **Conf** sends data sent such that only the intended receiver can read it.
- **Sec** is the conjugation of the Auth and the Conf modules.
- **Strong Sec** is the conjugation of the Strong Auth and the Conf modules.

Fig. 5 shows how to model the authenticated message sent by `User` to `Board`.

4.3 Semantics of the Security Modules

In GSD contracts are attached to the security modules on the Abstract Global level, describing the results of using the different modules. This can be used to define module semantics and to verify different implementations of a module by checking the specified security properties against the respective implementations.

Before defining the contracts, we briefly discuss the use of integrity in GSD. Integrity can have different meanings depending on the field it is being used in, and can even have slightly different definitions in the same field. Here, we consider integrity to mean that a message is not corrupted over time or in transit [14]. For message exchange, integrity is guaranteed when the contents of a message cannot be changed in transit without changes being instantly observable by the recipient. In the GSD framework we assume integrity in all exchanged messages, *e.g.*, by sending a signed digest of the message together with the full message. With integrity, we have the following rule (where $P\ sees\ m$ means that any principal is able to see m):[2]

$$\frac{X \rightarrow Y : m}{P\ sees\ m}$$

We can now present the rules for the different security modules. Please note that for the sake of space, we only present the more simple Auth and Sec modules.

Authentication. When a principal sees $Auth(X, w)$, he knows that the message was sent by X, but knows nothing about the freshness of the message:

$$\frac{Z\ sees\ Auth(X, w)}{Z\ sees\ w, Z\ believes\ X\ said\ w}$$

Confidentiality. A message that is confidential to X can only be read by him:

$$\frac{Z\ sees\ Conf(X, w), \quad Z\ is\ X}{Z\ sees\ w}$$

Security. The security module combines authentication and confidentiality:

$$\frac{Z\ sees\ Sec(V, X, w), \quad Z\ is\ X}{Z\ sees\ w, Z\ believes\ V\ said\ w}$$

Applying the rules presented above to the conjugation of the authentication and the confidentiality modules results in the same beliefs that were presented above for the security module:

$$\cfrac{X \rightarrow Y : Conf(Y, Auth(m))}{\cfrac{P\ sees\ Conf(Y, Auth(X, m))}{\cfrac{Y\ sees\ Conf(Y, Auth(X, m)),\ Y\ is\ Y}{\cfrac{Y\ sees\ Auth(X, m)}{Y\ sees\ m,\ Y\ believes\ X\ said\ m}}}}$$

[2] We extend BAN and SVO logics with principal variables P that represent all the principals that see the messages being exchanged.

5 The Concrete Levels

There are two concrete levels in GSD: the Concrete Global level (Sec. 5.1) and the Concrete Endpoint level (Sec. 5.2). The model of the system on these levels is closer to the languages used by the verification tools and the implementation, but not as simple and intuitive as the one in the Abstract Global level.

5.1 Concrete Global Level

We obtain the Concrete Global level by unfolding the different modules at the Abstract Global level. The different plugins for each of the different security modules and represent implementations of the corresponding security module, for example, implementations using TLS, WS-Security, or a Public Key Infrastructure (PKI). As previously mentioned, it is possible to verify the chosen implementation for a security module by checking that it satisfies the respective contract.

From this level the GSD framework interfaces with the Beliefs and Knowledge tool (Sec. 6.1) and the Open-Source Fixed-Point Model-Checker (Sec. 6.3). s.

For our example system we choose to implement the Auth module using a PKI infrastructure. The result of applying that plugin to the Auth module is shown in Fig. 6.

5.2 Concrete Endpoint Level

In order to translate from the Concrete Global level to the Concrete Endpoint level, we apply an endpoint projection [15]. This technique extracts the views of the different principals present in a specification of the global view of the system. The resulting model at this level has the views of the different principals that participate in the communication system.

This translation is performed by going through the model with a global view of the system and, for each of the actions in the model, generating the correspondent actions that are performed by the different principals. For example,a message being sent from `A` to `B` in the global view, is projected to the independent specification of the correspondent actions of `A` and `B`, *i.e.*, `A` would send the message and `B` would receive it.

The model of our message board example at this level is shown in Fig. 7.

6 Verification Tools and Code

In this section we present the formal methods tools that GSD currently interfaces with. The Beliefs and Knowledge tool (Sec. 6.1) was developed as part of the

```
User → Board: User, Encryption(message, PrivKey(User));
```

Fig. 6. An authenticated message of the example system in the Concrete Global level

```
        User:
            send(Board, (User,Encryption(message, PrivKey(User))))
        Board:
            receive(User,(User, Encryption(message, PrivKey(User))))
```

Fig. 7. An authenticated message of the example system in the Concrete Endpoint level

- **A $\longrightarrow$ B : m $\Longrightarrow$ P Received m** - When a message is sent between two principals, every principal with access to the Ether will receive that message.
- **A Received el $\Longrightarrow$ A Sees el** - If a principal receives an Element, he is able to see it (note that the principal might be able to see the Element, but not what is inside it).
- **A Sees Enc(el,privKey(P))** $\wedge$ **A Sees pubKey(P)** $\Longrightarrow$ **A Sees el** $\wedge$ **A Believes P Said el** - If principal A sees a message encrypted with another principal's private key and if A has access to the correspondent public key then A can see the encrypted element and also knows who sent it.

Fig. 8. Examples of the systems predefined rules

GSD framework. The GSD framework outputs the system model to code by replacing the different elements of the Concrete Endpoint specification with predetermined Java blocks implementing those elements.

6.1 The Beliefs and Knowledge Tool

The Beliefs and Knowledge tool (BAK) verifies the security of communication protocols by reasoning about the beliefs and the knowledge that the different principals involved in a communication system acquire throughout message exchange. The tool uses the Z3 SMT Solver [16] and adds an extra layer that facilitates the modeling of message exchanges and the reasoning about those messages.

Predefined System Rules. The extra layer defined in the BAK tool is composed of predefined system and inference rules specifying how principals construct and read the exchanged messages, how they acquire the different beliefs, etc. Some examples of these rules are shown in Fig. 8. A and B represent specific principals, P represents any principal, m represents a message in plaintext, $Enc(el, k)$ represents the encryption of the element el with the key k, and $pubKey(x)$ and $privKey(x)$ represent the public and private keys of principal x.

The third rule of Fig. 8 specifies which beliefs a principal can infer from a message encrypted with a private key: if the principal knows the correspondent public key, then he is able to decrypt it, see the element that had been encrypted, and have assurances on which principal encrypted the element. In Fig. 9, that rule is presented in SMT-LIB2.

There are two inputs for the BAK tool: a model (M) of the system we want to analyse and the goals we want to verify. Given these, the tool tests each goal

```
(assert (! (forall ((x Principal) (w Principal) (el Element))
           (! (=> (and (Sees x (EncModule el (PrivKey w)))
                       (Sees x (PubKey w)))
                        (and (Sees x el)
                       (Believes x (Said w el)))
                  )
                  :pattern ((Sees x (EncModule el (PrivKey w))))
              )
            )
       :named privKeyDecryption)
)
```

Fig. 9. One of the system rules regarding decryption.

```
(MsgSent User Board (Conc User (EncModule message (PrivKey User))))
```

Fig. 10. An authenticated message of the example system modeled in SMT-LIB2

(*goal*) against the modeled system. One implication of the way SMT works and the way we are modelling the system rules and the system itself is that we need to test the goals in the negated form. Both in the set of predefined system and inference rules (R) and in the system model (M) we only assert positive facts. When testing a goal in the positive form, the SMT solver will always find a model where the goal would be satisfiable since there will never be a negative rule to contradict the goal in the positive form. On the other hand, when testing a goal in the negative form it might contradict one of the assertions that are derived from $R \wedge M$, which can be interpreted as all the knowledge, *i.e.*, assertions, that can be derived from the system model. In that case the result will be unsatisfiable. If the negated goal does not contradict any of the assertions that are derived from $R \wedge M$, then the result will be satisfiable.

Therefore, we use $R \wedge M \wedge (\neg goal)$ to verify the system. If the system is satisfiable, then there is a representation of $R \wedge M$ where $\neg goal$ holds, which tells us that *goal* is not an assertion derived by $R \wedge M$. Due to the way we model the system and the system rules, this means that *goal* does not hold in the system. On the other hand, if the system is unsatisfiable, then there is no interpretation of $R \wedge M$ where $(\neg goal)$ holds. That can only happen if *goal* is derived from $R \wedge M$, which means that *goal* holds in the system. So, if any of the original goals we want to test is in the positive form we negate it before performing the test and interpret the result given by the tool according to that.

Fig. 10 shows the model of the authenticated message in SMT-LIB2, a standard format accepted as input by several SMT solvers, including Z3.

Tool Outputs. When analyzing $R \wedge M \wedge (\neg goal)$, the BAK tool does not only return satisfiability but also provides extra information that helps understanding and analysing the obtained results. If the system and the goal being analysed are satisfiable, then the tool also returns the representation that satisfies the assertions. On the other hand, if system and goal are unsatisfiable, the tool returns the unsatisfiability core, *i.e.*, a small set of assertions that make the system

```
unsat (privKeyDecryption)
```

Fig. 11. Output of the BAK tool for the example system.

```
<User, Board, User, {|message|}:K_User->. 0
|
(Board, User; board1, board2).decrypt board2 as {|;message|}:K_User+ in 0
```

Fig. 12. The automatically generated LySa code of the authenticated message in the example system

unsatisfiable. This set is not guaranteed to be minimal, but it provides useful information regarding the system and the analysis result. Extracting information from a satisfiable model is not as simple as extracting information from the unsatisfiability core since the model tends to be complex and not easily readable.

For the example system the implementation of our authentication message should give guarantees regarding the authenticity of the message. It is possible to test this by verifying that `Board` knows that the `message` was sent by `User`, which is specified as `Board believes (User said message)` in belief logic. The result of applying the BAK tool to this goal in the negative form is shown in Fig. 11.

The first line tells us that the system model together with that negated goal is unsatisfiable, which mean that the goal we wanted to verify holds. The second line of the output is the unsatisfiability core returned by Z3. It is the name of the rule shown in Fig. 9 and it tells us that the `Board` obtained the belief specified in the goal by decrypting the `message`.

6.2 LySatool

The LySatool [17] performs security analyses of protocols described in LySa [18]. The tool performs a static analysis of the LySa specification of the protocol in the presence of a Dolev-Yao attacker [13]. The LySatool is implemented in the Standard ML (SML) functional programming language and it starts by encoding the analysis into a proper constraint language and then uses Succinct Solver [19] to compute the least solution to those constraints.

The LySa code that is generated by our framework is shown in Fig. 12.

6.3 The Open-Source Fixed-Point Model-Checker

The Open-Source Fixed-Point Model-Checker (OFMC) [20] is a symbolic security protocol analyser that detects attacks on the protocol and performs a bounded session verification by exploring the transition system of the protocol representation. Its primary input language is the Intermediate Format (IF) [21] specification, which describes a security protocol as an infinite-state transition system using set rewriting. The tool also accepts AnB [22] as input, a language similar to Alice and Bob notation, which is then automatically translated to IF, defining a formal semantics for AnB in terms of IF. OFMC uses several

```
User → Board : User,{message}inv(pk(User));
```

Fig. 13. The automatically generated AnB code of the authenticated message in the example system

techniques that significantly reduce the search space of a protocol without introducing, or excluding, any attacks. Two of the major used techniques are *lazy intruder* and *constraint differentiation*. The first is a symbolic representation of the intruder while the latter is a general search-reduction technique. In Fig. 13, one can see the part of AnB code that corresponds to our authenticated message.

7 Evaluating GSD on ADS-B

In a recent evaluation, GSD was used to model and verify the Automatic Dependent Surveillance-Broadcast (ADS-B) [3] system [23], an air traffic management surveillance system that is being deployed with the intent of replacing the radar as the main system for airspace traffic management. This section first introduces the current implementation of ADS-B and then presents the way GSD was used to verify ADS-B.

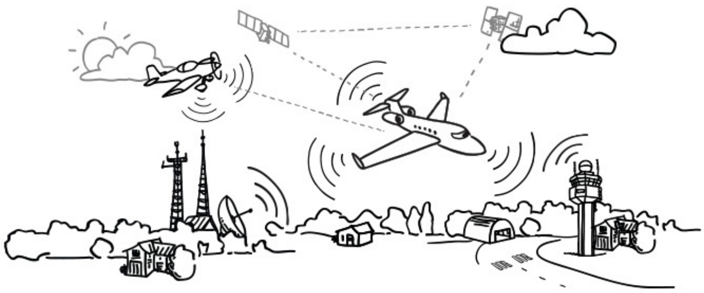


Fig. 14. Overview of the ADS-B system

7.1 The ADS-B System

ADS-B is a large wireless network, composed by ground stations and aircrafts that communicate with each other: the aircrafts report flight information (such as their position, velocity, and intent) and receive traffic and other information from the ground stations, as shown in Fig. 14. The main benefit of ADS-B is the provided higher accuracy regarding the aircraft position, which is crucial in an airspace where the aircraft density is increasingly higher.

[3] ADS-B General Information, `http://www.faa.gov/nextgen/implementation/portfolio/trans_support_progs/adsb/general/`, last accessed May 2014.

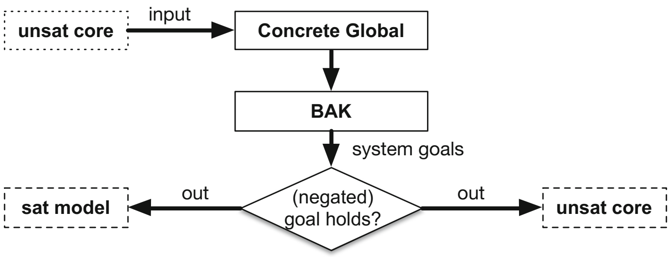


Fig. 15. The GSD framework applied to ADS-B

The legitimate agents taking part in the ADS-B communication system are the aircraft and the ground-stations. ADS-B has two components that allow these agents to communicate: ADS-B Out and ADS-B In. ADS-B Out consists of the messages that are broadcasted by the aircraft and ADS-B In concerns the capability of receiving the ADS-B Out messages. A message contains the aircraft's position and speed (both acquired through a positioning system, presently GPS) and potentially other information, such as intent. These broadcasted messages are received by the ground-stations and, in case ADS-B In is being used, the former will also be received by the aircraft that are within range of the broadcaster aircraft. As explained above, an aircraft is only capable of receiving air-to-air ADS-B messages broadcasted from other aircraft if it has equipment that provides ADS-B In capabilities. Another part of ADS-B In is the information broadcasted by the ground-stations. This will consist of traffic and weather information. During the transitional phase, the traffic information will have a mixture of ADS-B and Radar information, enabling the ADS-B In equipped aircraft to have a full view of the airspace surrounding it.

7.2 Applying GSD to ADS-B

The GSD framework was used to model and analyse the current implementation of ADS-B and its potential extensions as shown in Fig. 15. The most abstract level of the framework (Abstract Global) was not used, since that level is targeted for developing secure systems from scratch and not for modelling and analysing systems previously developed. Furthermore, the Concrete Endpoint level was not used either, since there was no interest in code, and the translation to the used tool to analyse our model is made from the Concrete Global level.

The ADS-B system model was used as input to the Concrete Global level, which was automatically translated into a language that can be used by the BAK tool (presented in Sec. 6.1) to verify the system and its properties (or goals), which are the other input to the framework.

GSD enabled the formal verification of the built-in security of the ADS-B communication system and reported that the system provides no authentication or confidentiality. GSD also enabled the security verification of the extensions suggested by Valovage *et al.* [24,25] and, in this case, it reported that authentication and confidentiality were provided in their respective extensions.

8 Conclusion

The Guided System Development framework aims at helping developers building secure communication systems. It does so by enabling the modelling of systems in a simple and intuitive language, its verification by connecting that model to different formal verification tools, and translating it to code. In this paper we presented the capabilities of the GSD Framework by discussing the process of modelling, verifying, and implementing an authenticated broadcasted message. We also introduced the Beliefs and Knowledge tool, which extends belief logics with explicit attacker knowledge and uses the Z3 SMT Solver to enable the verification of security properties of communication systems. Furthermore, we presented an evaluation of a new airspace navigation system and its proposed extensions using GSD to model and verify the system's communications.

We believe that this framework represents a big step towards closing the gap between systems development and verification, but more work is necessary: We aim at extracting more information from the satisfiability model return by the BAK tool in order to provide more complete feedback to the developer, finalising the interfaces with LySatool and OFMC, and optimising the overall tool integration so that it is easier to use for the system developers. We also work on interfacing to and integrating the results from more analysis tools.

Acknowledgements. We would like to thank Roberto Vigo, Sebastian Mödersheim (both from the Technical University of Denmark), and Kristin Y. Rozier (from NASA Ames Research Center) for many fruitful discussions.

References

1. Quaresma, J., Probst, C.W., Nielson, F.: The Guided System Development Framework. In: Pettersson, P., Seceleanu, C. (eds.) Proceedings of the 23rd Nordic Workshop Programming Theory, Västerås, Sweden, pp. 69–72 (October 2011)
2. Quaresma, J.: On Building Secure Communication Systems. PhD thesis, Technical University of Denmark (2013)
3. Gao, H., Nielson, F., Nielson, H.: Protocol Stacks for Services. In: Foundations of Computer Security (2009)
4. Song, D., Perrig, A., Phan, D.: Agvi—automatic generation, verification, and implementation of security protocols. In: Berry, G., Comon, H., Finkel, A. (eds.) CAV 2001. LNCS, vol. 2102, pp. 241–245. Springer, Heidelberg (2001)
5. Song, D.X., Berezin, S., Perrig, A.: Athena: A novel approach to efficient automatic security protocol analysis. Journal of Computer Security 9(1), 47–74 (2001)
6. Bhargavan, K., Fournet, C., Gordon, A.D., Tse, S.: Verified interoperable implementations of security protocols. ACM Transactions on Programming Languages and Systems (TOPLAS) 31(1), 5 (2008)

7. Blanchet, B.: An efficient cryptographic protocol verifier based on prolog rules. In: Proceedings of te 14th IEEE Computer Security Foundations Workshop, pp. 82–96 (2001)
8. Swamy, N., Chen, J., Fournet, C., Strub, P.Y., Bhargavan, K., Yang, J.: Secure distributed programming with value-dependent types. In: Proceeding of the 16th ACM SIGPLAN International Conference on Functional Programming, ICFP 2011, pp. 266–278. ACM, New York (2011)
9. Cade, D., Blanchet, B.: From computationally-proved protocol specifications to implementations. In: 2012 International Conference on Availability, Reliability and Security (ARES), pp. 65–74. IEEE (2012)
10. Blanchet, B.: A computationally sound mechanized prover for security protocols. IEEE Transactions on Dependable and Secure Computing 5(4), 193–207 (2008)
11. Burrows, M., Abadi, M., Needham, R.: A logic of authentication. ACM Trans. Comput. Syst. 8, 18–36 (1990)
12. Syverson, P.: A unified cryptographic protocol logic. Technical report, DTIC Document (1996)
13. Dolev, D., Yao, A.: On the security of public key protocols. IEEE Transactions on Information Theory IT-29(2), 198–208 (1983)
14. Cullen, C.T., Hirtle, P.B., Levy, D., Lynch, C.A., Rothenberg, J.: Authenticity in a digital environment (2000)
15. Carbone, M., Honda, K., Yoshida, N.: Structured communication-centred programming for web services. In: De Nicola, R. (ed.) ESOP 2007. LNCS, vol. 4421, pp. 2–17. Springer, Heidelberg (2007)
16. de Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008)
17. Buchholtz, M.: User's Guide for the LySatool version 2.01. DTU (April 2005)
18. Bodei, C., Buchholtz, M., Degano, P., Nielson, F., Nielson, H.: Static validation of security protocols. Journal of Computer Security 13(3), 347–390 (2005)
19. Nielson, F., Riis Nielson, H., Sun, H., Buchholtz, M., Rydhof Hansen, R., Pilegaard, H., Seidl, H.: The succinct solver suite. In: Jensen, K., Podelski, A. (eds.) TACAS 2004. LNCS, vol. 2988, pp. 251–265. Springer, Heidelberg (2004)
20. Mödersheim, S., Viganò, L.: The open-source fixed-point model checker for symbolic analysis of security protocols. In: Aldini, A., Barthe, G., Gorrieri, R. (eds.) FOSAD 2007/2008/2009. LNCS, vol. 5705, pp. 166–194. Springer, Heidelberg (2009)
21. AVISPA: Deliverable 2.3: The intermediate format (2003), `http://www.avispa-project.org`
22. Mödersheim, S.: Algebraic Properties in Alice and Bob Notation. In: 2009 International Conference on Availability, Reliability and Security (ARES), pp. 433–440. IEEE (2009)
23. RTCA: DO-242A: Minimum Aviation System Performance Standards for Automatic Dependent Surveillance Broadcast (ADS-B). Technical report, RTCA (2002)
24. Valovage, E.: Enhanced ADS-B Research. In: 2006 IEEE/AIAA 25th Digital Avionics Systems Conference, pp. 1–7 (October 2006)
25. Viggiano, M., Valovage, E., et al.: Secure ADS-B Authentication System and Method (October 12, 2007), WO Patent 2,007,115,246

Processes and Data Integration in the Networked Healthcare
(Track Introduction)

Tiziana Margaria[1] and Christoph Rasche[2]

[1] Chair of Service and Software Engineering, University Potsdam, Germany
margaria@cs.uni-potsdam.de
[2] Chair of Management, Professional Services and Sports Economics, University Potsdam, Germany
christoph.rasche@uni-potsdam.de

Forward-looking issues in the Information and Communication Technology (ICT) for healthcare and medical applications include process integration, data annotation, ontologies and semantics, but also any automatization approach that is not imposed from the IT experts (in-house support or external consultants) but instead allows more flexibility and also a more direct ownership of the processes and data by the healthcare professionals themselves. In the ISoLA-Med workshop in Potsdam in June 2009 and in this track at ISoLA 2012 we showed that a set of innovative research topics related to the future of healthcare systems hinge on the notion of simplicity, for both the end users and the designer, developers, and to support change management and the agility of evolution.

Current hot issues that are expected to shape the competitiveness of the European ICT in the next few decades and which require investigation from the perspective of simplicity in IT at the networked system level revolve around the notion of simplicity [2] and its elevation to a design paradigm including:

– Balancing IT-aspirations with user demands: How to bridge the widening gap between software engineers and front-end clients.
– From sophisticated to smart technologies: User empowerment through simplicity, manageability, adaptability, robustness, and target group focus.
– Handing over IT power to the co-value creating customers: Users as process designers, owners and change agents.
– Competing for the future: Sketching viable IT-roadmaps for multiple strategies.

In this track, we consider three facets of the IT effects on the healthcare sector that show a high potential for IT-based process support and optimization:

– *Simple Management of High Assurance Data in Long-lived Interdisciplinary Healthcare Research: A Proposal* [4] describes the data management needs of a large interdisciplinary research project coordinated at the Cancer Metabolism Research Group in the Institute of Biomedical Sciences at USP in Brazil. There, the central issue is which kind of IT can allow the healthcare

specialists to model, govern, and manage complex data landscapes, considering that in such a research context the production, reliability, and long term availability of high quality and high assurance data are of vital importance. The authors sketch how the combination of DyWA and jABC provides a foundation for meeting those needs.
– *Domain-specific Business Modeling with the Business Model Developer* [1] presents a tool and the underlying framework for the creation of domain-specific business models, and its application in a consortial project concerning business models for Personalized Medicine. The few existing business modeling tools do not allow analysis and essentially mimic pencil and paper approaches. In contrast, the BMD allows a domain-specific library of parametric model components with declared areas of applicability, that serves as a basis for the development of custom techniques for business model analysis.
– *Dr. Watson? Balancing Automation and Human Expertise in Healthcare Delivery* [3] (short paper) describes how to identify process design changes that support the integration of new information technologies into the healthcare delivery process. It is illustrated on a technology-based remote diagnosis of radiology images to address the increasing demand for the reading of mammograms.

References

1. Boßelmann, S., Margaria, T.: Domain-specific business modeling with the business model developer. In: Margaria, T., Steffen, B. (eds.) ISoLA 2014, Part II. LNCS, vol. 8803, pp. 545–560. Springer, Heidelberg (2014)
2. Margaria, T., Steffen, B.: Simplicity as a Driver for Agile Innovation. IEEE Computer 43, 90–92 (2010)
3. Gaynor, M., Wyner, G., Gupta, A.: Dr. Watson? Balancing automation and human expertise in healthcare delivery. In: Margaria, T., Steffen, B. (eds.) ISoLA 2014, Part II. LNCS, vol. 8803, pp. 561–569. Springer, Heidelberg (2014)
4. Margaria, T., Floyd, B.D., Camargo, R.G., Lamprecht, A.-L., Neubauer, J., Seelaender, M.: Simple management of high assurance data in long-lived interdisciplinary healthcare research: A proposal. In: Margaria, T., Steffen, B. (eds.) ISoLA 2014, Part II. LNCS, vol. 8803, pp. 526–544. Springer, Heidelberg (2014)

Simple Management of High Assurance Data in Long-Lived Interdisciplinary Healthcare Research: A Proposal

Tiziana Margaria[1], Barry D. Floyd[2], Rodolfo Gonzalez Camargo[3,5], Anna-Lena Lamprecht[1], Johannes Neubauer[4], and Marilia Seelaender[3]

[1] Chair Service and Software Engineering, University of Potsdam, Germany
{margaria,lamprecht}@cs.uni-potsdam.de

[2] Orfalea College of Business, California Polytechnic State University, USA
bfloyd@calpoly.edu

[3] Cancer Metabolism Research Group, Institute of Biomedical Sciences, University of São Paulo, Brazil
rodolfogcamargo@usp.br, seelaend@icb.usp.br

[4] Chair Programming Systems, TU Dortmund, Germany
johannes.neubauer@cs.tu-dortmund.de

[5] Chair Biochemistry of Nutrition I, Institute for Nutritional Sciences, University of Potsdam, Germany
rogonzal@uni-potsdam.de

Abstract. Healthcare research data is typically produced, curated, and used by scientists, physicians, and other experts that have little or no professional affinity to programming and IT system design. In the context of evidence-based medicine or translational medicine, however the production, reliability, and long term availability of high quality and high assurance data is of paramount importance. In this paper we reflect on the data management needs we encountered in our experience as associated partners of a large interdisciplinary research project coordinated at the Cancer Metabolism Research Group, Institute of Biomedical Sciences at University of São Paulo in Brazil. Their research project involves extensive collection of detailed sample data within a complicated environment of clinical and research methods, medical, assessment, and measurement equipment and the regulatory requirements of maintaining privacy, data quality and security.

We use this example as an illustrative case of a category of needs and a diversity of professional and skills profiles that is representative of what happens today in any large scale research endeavor. We derive a catalogue of requirements that an IT system for the definition and management of data and processes should have, how this relates to the IT development and XMDD philosophy, and we briefly sketch how the DyWA + jABC combination provides a foundation for meeting those needs.

1 Introduction

Evidence based medicine [2] informs the therapeutic decisions through knowledge derived from research and patient-related clinical data. In this approach the knowledge of the single specialist and healthcare professional plays a role in guaranteeing the best possible decisions of patient care, and the individual knowledge and experience is enhanced by the collective experience of similar cases. The quality and reliability of medical evidence therefore is crucially connected with the availability and the accessibility of high-quality collections of data. This is particular critical for rare conditions; fortunately there are registers of such conditions in place. At the European level, for example, the EPIRARE platform [1] is a European platform for rare disease repositories that aims at coordinating the effort at the European level connected with the existing national level of rare disease repositories. These platforms have the goal of defining the needs of the EU registries and databases on rare diseases, identifying key issues to prepare a legal basis, agreeing on a common data set and elaborate procedures for quality control, as well as on the register and platform scope, governance and long-term sustainability.

In the USA, the NIH maintains a registry of patients and a data repository especially for rare diseases (GRDR) [3]. Citing their homepage, *"The goal is to establish a data repository of de-identified patient data, aggregated in a standardized manner, using Common Data Elements (CDEs) and standardized terminology. This data will be available to all investigators to enable analyses across many rare diseases and to facilitate various biomedical studies, including clinical trials, in pursuit of developing drugs and therapeutics to improve the healthcare and the quality of life for the many millions of people who are diagnosed with rare diseases. De-identification of patient's data will utilize the Global Unique Identifiers (GUID) system which could also link patient's data to bio-specimen data set."*

The issues central to research, translational medicine, and evidence-based medicine are closely related with the questions of data collection, maintenance, and availability for use in the scientific processes and in patient care. Reuse of data cannot happen without adequate IT support, and it cannot be pervasive nor sustainable if the use, maintenance, and evolution of the data and the processes are kept culturally separated from the scientists and medical professionals that produce, curate, and use that data. The largest barrier to adoption and "ownership" of data and process management is the illiteracy or semi-literacy of these highly skilled and motivated user groups in IT related matters and data management in particular.

Accordingly, in this paper, we discuss the task of medical research and the need for easy-to-use process and data modeling tools in a multi-dimensional, heterogeneous environment. This need is conjoined with the impelling need for high process and data quality. We illustrate our points on a specific case study concerning a consortium of researchers studying the systemic inflammation in cachectic cancer patients, a condition that is not rare, but so far under-studied. In particular, we resort to a before/after illustration style, providing direct evidence of the different levels of formalization we are achieving once the proper IT means are introduced.

The paper is structured as follows. Section 2 introduces the case study and illustrates the diversity of knowledge, heterogeneity of expertise and goals as well as the geographic distribution of the participating research groups. Section 3 discusses the needs and characteristics of proper data management. The current situation is illustrated in Section 4. Section 5 introduces the data modeling in DyWA and finally Section 6 summarizes our discussion and perspectives.

2 A Translational Medicine Research Project: Fighting Cachexia in Cancer Patients

The Cancer Metabolism Research Group at the Institute of Biomedical Sciences, University of São Paulo, São Paulo, Brazil, in conjunction with medical staff at the University Hospital at the University of São Paulo and other researchers located in Spain, Italy, USA and Germany are actively engaged in conducting clinical trials to understand the dynamics of cachectic cancer. Cachexia is defined in [9] as "*a multifactorial syndrome defined by an ongoing loss of skeletal muscle mass [...] that cannot be fully reversed by conventional nutritional support and leads to progressive functional impairment*". Cancer cachexia is frequently under-diagnosed due to a large variety of symptoms and effects that make it difficult to recognize in the early stages of the disease. It is therefore under-treated until a late, sometimes terminal, status is reached. This research project aims at facilitating early diagnosis and treatment by establishing practical criteria that are applicable in clinical practice. Which parameters to observe and which thresholds to use are, in fact, still open questions. There is increasing consensus that systemic inflammation may be related to the onset of cachexia, so inflammation-based markers are being studied as a means of diagnosis and prognosis. There is growing evidence from studies on humans and animals [4] that by acting on the inflammation, cachexia may be attenuated.

Thus, this research project[1] involves extensive collection of detailed sample data within a multidiscipinary environment of clinical and research methods, medical, assessment, and measurement equipment and the regulatory requirements of maintaining privacy, data quality and security. Currently, in this project the research staff has a focus on medical excellence and analysis, not on data management and security.

2.1 How the Project Is Organized

The main goal of the research is to study at the molecular level resident/recruited cell interaction and relative contribution to local (and systemic) inflammation

[1] Project "Systemic Inflammation in Cachectic Cancer Patients: Mechanisms and Therapeutical Strategies. A Translational Medicine Approach", funded by FAPESP (2013-2017).

in the adipose tissue, skeletal muscle, liver and tumor of cancer patients in an attempt to describe patterns that lead to the onset, maintenance and aggravation of cachexia.

A second goal is to propose and test therapeutic strategies to counteract chronic system inflammation in cachexia. With this purpose, a protocol of patient and control group physical exercise has been established for chronic submaximal intensity endurance exercise. Measuring the effects of this training period is thus a further goal of the project.

The main challenges from an IT point of view are the heterogeneities and the diversity, with the following two main aspects.

2.2 Interdisciplinarity: Rich Life Science Expertise, Minimal Information Technology (IT) Knowledge

This study on cachexia cancer is a highly interdisciplinary project where teams of researchers with very disparate backgrounds cooperate in an effort to holistically understand the mechanisms underlying this devastating condition. The represented disciplines include: molecular biology, biochemistry, physiology, psychology, oncology, surgery, psychology, radiology, nutrition sciences, and kinesiology. As we see, the medical and life science skills are strongly represented. The IT side is represented by the collaboration of a knowledge representation colleague at USP and external support by T. Margaria's group in Potsdam.

2.3 Geographic Distribution: The Different Groups Participating, Medical and IT

This project spans also different geographical areas additionally to the different organizational affiliations and differences in expertise and tasks. The core groups focusing on medical research activities are located in Brazil, in a collaboration among different Institutes of USP, the USP University Hospital and UNIFESP in São Paulo, and the Universidade de Mogi das Cruzes (UMC). However, a second layer of research groups cooperates remotely in a worldwide range on topics as shown in Table 1.

IT expertise is externally contributed by technical specialists at the University of Potsdam in Germany and at California State Polytechnic University in the USA. It is possible, and actually hoped for, that in the course of the project other groups with medical skills and/or technical expertise will join this collaboration. Such an extension would be an excellent indication of the relevance and quality of the research. We expect that the additional groups will pursue their own research goals, but be interested in accessing subsets of the data that is currently being collected. Such a perspective brings us to the issue of the *long term availability and curation* of the data set and the documentation of the data collection, current analyses and results.

The following two tables provide background information about the location and affiliations of the project participants. Table 1 focuses on all the current

Table 1. Remote associated groups and their geographical location and affiliation

Group	**Institute/Country**
Alessandro Laviano, Maurizio Muscaritoli	Department of Clinical Medicine Sapienza University of Rome, Rome, Italy.
Giorgio Trinchieri, Romina Goldzmid	Center for Cancer Research National Cancer Institute, Bethesda, Maryland USA
Josep M. Argilés, Silvia Busquets	Cancer Research Group, Institut de Biomedicina, Univ. Barcelona Barcelona, Spain
Nicolaas Deutz	Department of Health & Kinesiology Texas A&M University Bryan, Texas, USA
Stephen Farmer	Department of Biochemistry Boston University School of Medicine Boston, MA, USA
Gerhard Paul Püschel	Institute of Nutritional Science University of Potsdam Potsdam, Germany
Tiziana Margaria	*Institute of Informatics University of Potsdam Potsdam, Germany*
Barry D. Floyd	*California State Polytechnic University San Luis Obispo, CA, USA*

remote participants, medical and IT, while Table 2 focuses on the medical researchers and provides details about their specialties and medical research foci.

In this articulated context, a proper data management, elastic and sustainable, made for growth, is a clear wish. Fortunately, there is no legacy with which we need to maintain compatibility.

3 Needs of Proper Data Management

Data management is critically important in medical research projects, because any tainting or imprecision can undermine trust. Individual treatments, government policies, and societal initiatives among many other aspects critically depend on the correctness of outcomes based on collected and interpreted data, and high quality data requires a consistent understanding and application of effective processes and controls. In modern experimental science, this happens with the help of IT. The simplified data life cycle as shown in Fig. 1 encompasses collection, storage, publication, access/update, maintenance/integration, and archive/destruction. Our current emphasis in this project is on developing an infrastructure that maintains the data and provides appropriate access to medical researchers over a long term and still largely unknown use of these data.

Table 2. Overview of the main participating medical research groups

Group Leader	**Location**	**Species**	**Questionnaires**	**Anthropometrics**	**Tissues**	**Cells**	**Proteins and Genes**	**Lipid profile**	**Lipodomics**	**Faeces**
Marilia Seelander	Biochem. USP , São Paulo (BRA)	human (cancer), rat (cancer)	Cancer Quality of life Questionnaire Core-30, Karnofsky-Index	height, weight, weight loss, handgrip	liver, intestine, stomach, blood, tumor, adipose tissue, muscle	Immune cells, Adipocytes, Fibroblasts	cytokines, albumin, acute phase proteins, intracellular proteins, micro RNA	HDL, LDL, Total cholesterol, Triglycerides, Free fatty acids	Saturated fatty acids, Mono unsaturated fatty acids, Poly unsaturated fatty acids, Prostaglandin, Lipid mediators	Bacterial DNA
Miguel Luis Batista Junior	Univ. Mogi das Cruzes, (BRA)	mice (cancer)			adipose tissue		cytokines, intracellular proteins			
Claudia Oller, Lila Missae	UNIFESP, São Paulo (BRA)	human (cancer)			tumor, adipose tissue, blood		cytokines, intracellular proteins			
Alessandro Laviano, Maurizio Muscaritoli	Medicine, La Sapienza, Rome (ITA)	human (cancer)	anorexia questionnaire	height, weight, weight loss, handgrip						
Giorgio Trinchieri, Romina Goldzmid	NIH, Bethesda, MA (USA)	human (cancer)								bacterial DNA
Josep M. Argilés, Silvia Busquets	Barcelona (E)	human (cancer)	the cachexia score (CASCO)							
Nicolaas Deutz	Texas A&M University Texas (USA)	human			adipose tissue, muscle		fractional synthesis rate			
Gerhard Püschel	Nutritional sciences, Univ. Pots (D)	rat, mouse			liver, adipose tissue		cytokines, intracellular proteins		prostaglandins	

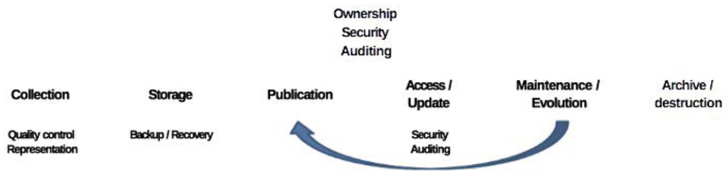


Fig. 1. Standard data life cycle in scientific data management

It is therefore organized in an agile fashion that allows ease of use, integration and evolution as new data arises and new technologies, users, and user groups come into being.

We first summarize the essentials of the data management lifecycle, and then discuss briefly the process aspect of data manipulation and governance.

3.1 Lifecycle in Scientific Data Management

Data collection. Data management is expert knowledge intensive, requiring an understanding of which data needs to be *collected.* Raw, primary data should be collected as close to the source of the data as possible and at the least possible level of aggregation. For example, collecting a subject's weight and height is preferable to recording their BMI, which can be computed from these two measures. Where the data is collected and under which experimental or measurement conditions play important roles in putting effective collection techniques in place, as well how the data arrives and its arrival speed. Quality must be controlled from the inception, because it is mostly impossible a-posteriori to rectify incorrect or missing information. For instance, acceptable ranges of values (e.g., patient weight) and uniqueness constraints (e.g., patient email address) must be identified and, if possible, checked directly at collection time in order to make any corrections immediately at the source. The appropriate representations (e.g., text, numbers, pictures), i.e. in the IT world the data types must be identified and recorded, along with their units of measurements (e.g., mg). Similarly, any encoding (e.g. m/f for gender) must be identified and retained. This is part of the additional information (in IT terms, the metadata and environment description) needed to later understand, correctly interpret, and possibly establish the validity or usability of the data. Any change in an encoding (for example in Sweden a new gender value "X" was included), as well as in the experimental or measurement conditions (like changes in any equipment, solutions, or methods that are used to collect data or data samples), the date/time of the change must be stored. Such changes may in fact impact the suitability, precision, and correctness of subsequent analyses and interpretations.

Data representation includes also considering privacy issues: information about a patient's identification should be managed so that only those who have the right to know the association between the patient identity data (e.g., name, address, phone) and the patient study data (e.g., weight, tumor classification) have access to both sets of information. This means assigning a unique identifier to the patient and then using this patient id when storing patient study data. These two sets of data must then be managed. This can often mean using different physical data stores (e.g., in paper based systems the sets of paper must be stored in different locations). In computer based systems, including appropriate access controls can be effective. In some cases, both the data and the meta data must be controlled. In terms of privacy, letting inappropriate personnel know what data is being collected and thus allowing inferences to be conducted, can be invasive to the patient's privacy.

Data storage. Proper *storage* has come to include security aspects, to minimize the potential for data leakage beyond legitimate access. The location where data will be stored, the technical formats as well as potential data volumes play a role in establishing and maintaining security, reducing errors in any data movement (e.g., from hard copy to digital) as well as for any curation activity that might take place in the future. For example, data stored on older technologies or in application software that becomes unsupported may become unreadable over time; moreover, understanding that some data technologies fail after lengthy periods of time is important when determining where and how redundantly to store the data. Potential data loss through technology failures, natural or manmade disasters should be managed and mitigated against. This leads to policies for fault tolerance, system resilience, and the choice of appropriate backup and recovery technologies.

How data is best stored depends on the expected access and update pattern. How the data is to be used, by whom, and from where are all key aspects of the data storage design. For example, storing data in a spreadsheet provides a lower level of access and usability than in a professional database management system. Storing data on a USB-stick or on one laptop is less desirable because (1) sharing is hard (especially globally), (2) keeping track of where the data is located is difficult, (3) managing different versions of the data is time consuming and error prone, and (4) is very insecure, resulting in a high risk of data loss.

Governance. On the *governance* side, ease of access to the users must still be traded off against the wish to grant controlled access, i.e. given proper credentials. Setting up a secure system with appropriate safeguards and with an adequate role/rights management requires understanding which personnel will be given authorization to operate and in which way: input, view, change, or, potentially, also delete data. This access and permission control may be established at different levels of the data hierarchy, for example defining a fine granular permission system that individually consents e.g. viewing only certain fields/attributes of a record versus seeing or manipulating the entire record or potentially all of a complete file/table/object.

In the current project we foresee for the moment a data owner-centered management of data, meaning that the researcher, or unit, and/or possibly institution who creates the data is also the (only) instance who can manipulate it and possibly invalidate or delete it. We currently tend to not allow deletes, but just let obsolete or incorrectly entered data to be marked as invalid, in order to guarantee completeness of the data set. Other participating entities are allowed to view and read the data, e.g. in order to use it in computations or for classification purposes, but without any modifying rights (pure passive access).

Concomitantly, maintaining a record of any access to confidential data is essential. Such a logging scheme provides an indirect level of control simply by letting users know that all accesses are recorded. Many funding and oversight institutions mandate that such controls and records be maintained. These records must also be out of the reach of any of the database administrators and only allowed to be viewed / maintained by authorized, study personnel.

End of life. The end of the data life cycle is the archiving / destruction. This must be planned for in advance as well: If data is to be maintained for a long period of time, an appropriate long-term storage technology must be put into place. If the data is to be destroyed, all its images (e.g., currently used data stores, backup data stores, data which has been 'checked out' by study personnel and stored separately) must be identified and effectively erased using current state of the art technologies. Ideally, therefore, there should be also a register of the images, that tracks who and when they were made and thus how many of them are currently in existence and who is responsible for each of them.

Security and auditing. As we see, the issues of ownership, security and auditing are actually overarching the entire lifecycle. Data ownership must be identified and assigned from the very beginning of the project. As the owner is responsible for any key decisions about data use and data access, managing ownership issues and rights of access and use is especially important if, like in our project, the data is expected to evolve and be used for other not currently specified applications. Often there are specific data collection and use agreements in place with any patients / subjects: they state how the data may be used and the owner must abide by these agreements. It is the owner who assigns rights to any legitimate project partner, and it is the owner's responsibility to ensure that the use will respect any usage restriction.

Thus, the security and auditing aspects discussed in the access/update stage actually pervade the entire life cycle and are a cross-cutting concern of global reach for the entire system. Data leakages happen in fact often inadvertently.

In this line of thoughts, it is natural to think that any action performed on the data, as well as its governance and analysis, are actually connected with a recipe how to properly do it. In the IT world, recipes are processes.

3.2 The Need of Proper Process Management

The management as well as the analysis of data are themselves processes. Actually, any handling that answers a "How" question is a process: it is an explanation of the (human and automated) steps that need to be done to transform data into any kind of results and outcome. We find in this research project for example processes that explain how data and samples are collected and treated, computation processes that transform the collected raw data into information (e.g. the computation of the BMI) or decisions (for example, how to perform the classification for the patient grouping described in Sect. 4). Data management and governance are also linked to processes: how do we define who can input, verify, correct, deactivate data, define or modify computational processes, perform statistical analyses? How do we grant, modify, and revoke access rights to data, information, and processes themselves?

These are all implicit requirements for the use of an integrated process modeling tool. Such a tool should easily manage the data, through processes that are domain-specific scientific workflows [15,14,17,16,18]. It should also manage and steer the entire system including the processes and the meta-data needed for the governance, that are instances of otherwise application-domain independent data management related processes. While scientific workflows describe how to carry out the experiments and analyses specific to this project, the generic data management-related processes describe in general e.g. how to log in and authenticate a user, how to define or change roles and rights, how to create periodic reports that summarize the status and monitor the health of the data and its access. They are simply instantiated and configured to define how to handle these tasks for this specific experiment and these specific participating researchers.

In an evolving project, ideally, we wish thus more than just a database: we need a flexible system where to design and evolve the data and the processes in an agile fashion, possibly putting these tasks in the hands of the researchers themselves. This excludes direct programming as well as the most traditional process and workflow management tools, because they are not integrated with the data definition and management layer, and require complex mappings between the two.

A model-based approach has the advantage of describing (instead of prescribing) the processes. State-of-the-art Model Driven Development (MDD) offers a number of advantages for early validation and verification of the behaviors at the model level, prior to coding, for example by model checking [6,22]. Additionally, eXtreme MDD [21] offers also the advantage of co-creation, co-design and co-evolution of the models also for domain experts like our researchers in the project, that are not IT experts. The key to this new level of accessibility is a domain specific language for the things (data and functionalities) that populate the models of the experiments and data analyses, combined with a mathematically and software technically precise and complete graphical definition of these behaviors in terms of workflows that orchestrate the operation of functionalities on those data items. The jABC [28,24,23] is a framework that supports this agile way of enacting the XMDD paradigm in a continuous model driven engineering fashion [20]. It delivers the full functionality of early validation and verification [11] as well as the executability via code generation [13].

4 How the Work Is Being Conducted to Date

The current data collection procedures include a variety of processes and data collection instruments. These include records created online by the participating hospital, pdf files of data analysis results created by participating labs, hardcopy survey instruments such as questionnaires, and MS Excel spreadsheets. Each of these technologies stems from the various partners associated with the project, and together they pose a challenge in such as disparate, diverse organizational setting. We illustrate the general situation on a subset of these data collection activities. We chose to focus on the collection of data with an emphasis on the use of MS Excel spreadsheets, that include data and also computations, and to illustrate some challenges and potential opportunities for improvements.

A key task in the research project is the classification of study participants (patients) into various control and experimental groups. The media used so far to collect the data about each patient and then assign him/her to the appropriate group include standard questionnaires like EORTC's QLQ-C30 [7] concerning quality of life, basic patient data records stored at the University Hospital, and other datasets that include anthropometric data like gender, age, and height.

To assess patient conditions over the course of the study, time series data such as weight (which is a very important measure for cachexia patients) is tracked. Other measurements and experiments concern the concentration of certain substances in the blood or in tissues, and their analysis with respect to DNA, RNA, or proteins.

In the course of the project, the raw data is gradually collected, classified, and then used in correlations and statistical analyses in order to find out significant patterns, dependencies, or similarities.

4.1 Example: Assigning Patient to Groups

The model for assigning patients into appropriate study groups was implemented in MS Excel, with data entry and its analysis all in one table – the typical situation spreadsheet users face. We use the definition of cachexia by Evans et al. [8] requiring a weight loss of at least 5% in 12 months or less, plus a positive assessment on at least three of the following criteria:

- Decreased muscle strength
- Fatigue
- Anorexia
- Low fat free mass index
- Abnormal biochemistry:
    - Increased inflammatory markers CRP (>5.0mg/l), Il-6 (>4.0pg/ml)
    - Anemia
    - Low serum albumin

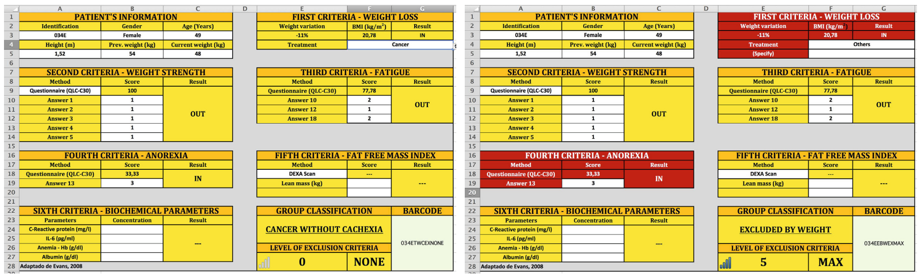
The spreadsheet models both the patient data along and the assessment data on these criteria.

4.2 Illustration of Spreadsheet Use

The first step is to define an identification ID for the patient (entered in cell A3) based on the last patient of the same project. Then, information on gender, age, height, previous and current weight is entered into cells in the upper right hand corner of the spreadsheet.

- Once this information is filled, the spreadsheet calculates the percentage of weight loss and the Body mass Index (BMI) as shown in the upper right hand corner, resulting in an assessment for the first criteria of percent weight loss.
- The second, third and fourth criteria are based on the answers given in the Quality of life questionnaire QLC-30 (validated in Portuguese).
- The fifth criterion is based on DEXA Scan analysis or MUAMA (mid upper arm muscle circumference) for age and gender (see [8] for details).
- The sixth criterion is based on serum analysis of biochemical parameters, which are performed by different groups and submitted via a PDF document to be entered into the spreadsheet.

Once the spreadsheet is filled, the model calculates the number of fulfilled criteria and identifies the appropriate group for the patient (e.g., cancer without cachexia or the control group).



	A	B	C	D	E	F	G
1	PATIENT'S INFORMATION				FIRST CRITERIA - WEIGHT LOSS		
2	Identification	Gender	Age (Years)		Weight variation	BMI (kg/m^2)	Result
3	034E	Female	49		-11%	20,78	IN
4	Height (m)	Prev. weight (kg)	Current weight (kg)		Treatment	Cancer	
5	1,52	54	48				
6							
7	SECOND CRITERIA - WEIGHT STRENGTH				THIRD CRITERIA - FATIGUE		
8	Method	Score	Result		Method	Score	Result
9	Questionnaire (QLC-C30)	100	OUT		Questionnaire (QLC-C30)	77,78	OUT
10	Answer 1	1			Answer 10	2	
11	Answer 2	1			Answer 12	1	
12	Answer 3	1			Answer 18	2	
13	Answer 4	1					
14	Answer 5	1					
15							
16	FOURTH CRITERIA - ANOREXIA				FIFTH CRITERIA - FAT FREE MASS INDEX		
17	Method	Score	Result		Method	Score	Result
18	Questionnaire (QLC-C30)	33,33	IN		DEXA Scan	---	---
19	Answer 13	3			Lean mass (kg)		
20							
21							
22	SIXTH CRITERIA - BIOCHEMICAL PARAMETERS				GROUP CLASSIFICATION		BARCODE
23	Parameters	Concentration	Result		CANCER WITHOUT CACHEXIA		034ETWCEXNONE
24	C-Reactive protein (mg/l)		---				
25	IL-6 (pg/ml)						
26	Anemia - Hb (g/dl)				LEVEL OF EXCLUSION CRITERIA		
27	Albumin (g/dl)				0	NONE	
28	Adaptado de Evans, 2008						

	A	B	C	D	E	F	G
1	PATIENT'S INFORMATION				FIRST CRITERIA - WEIGHT LOSS		
2	Identification	Gender	Age (Years)		Weight variation	BMI (kg/m^2)	Result
3	034E	Female	49		-11%	20,78	IN
4	Height (m)	Prev. weight (kg)	Current weight (kg)		Treatment	Others	
5	1,52	54	48		(Specify)		
6							
7	SECOND CRITERIA - WEIGHT STRENGTH				THIRD CRITERIA - FATIGUE		
8	Method	Score	Result		Method	Score	Result
9	Questionnaire (QLC-C30)	100	OUT		Questionnaire (QLC-C30)	77,78	OUT
10	Answer 1	1			Answer 10	2	
11	Answer 2	1			Answer 12	1	
12	Answer 3	1			Answer 18	2	
13	Answer 4	1					
14	Answer 5	1					
15							
16	FOURTH CRITERIA - ANOREXIA				FIFTH CRITERIA - FAT FREE MASS INDEX		
17	Method	Score	Result		Method	Score	Result
18	Questionnaire (QLC-C30)	33,33	IN		DEXA Scan	---	---
19	Answer 13	3			Lean mass (kg)		
20							
21							
22	SIXTH CRITERIA - BIOCHEMICAL PARAMETERS				GROUP CLASSIFICATION		BARCODE
23	Parameters	Concentration	Result		EXCLUDED BY WEIGHT		034EEBWEXMAX
24	C-Reactive protein (mg/l)		---				
25	IL-6 (pg/ml)						
26	Anemia - Hb (g/dl)				LEVEL OF EXCLUSION CRITERIA		
27	Albumin (g/dl)				5	MAX	
28	Adaptado de Evans, 2008						

Fig. 2. Patient Group Classification. Group evaluation for this patient a) record in case of cancer (see field G4), on the left, and b) in absence of cancer, on the right.

We see in Fig. 2 how the Patient Group Classification is carried out on a specific example, collecting and combining data items and information items provening from different exams and questionnaires. On the left we see the evaluation for this (assumed) patient in presence of a weight loss of 11% in the observed time frame and in presence of a cancer diagnosis (field G4 of the excel spreadsheet). As we see, most data concurring to the multidimensional classification stems from heterogeneous sources: patient demographics, height, blood values, weight difference, and selected well-being indicators from the standardized QLQ-C30 questionnaire. Several criteria combine data from different provenance, and the final grouping depends on a number of these criteria. Part of the classification algorithm, as implemented in the excel spreadsheet, is shown on the right in the input mask as original excel formula. The computed classification outcome is reported in field G23: cancer without cachexia. On the right we see the cachexia index evaluation for the same patient data but in absence of cancer: we see that now criteria 1 and 4 are not met anymore, and thus the corresponding subtables are now red, and we see that the group cachexia classification outcome in field G23 now excludes the patient.

4.3 Discussion on the Current Modeling Approach

The use of an MS Excel spreadsheet shows several shortcomings.The data entry process must be conducted twice, once at the source collection point and again into the model. Currently no data quality controls are implemented: syntactic checks of wellformedness (e.g. data range constraints) are missing. Importantly, the current design allocates one spreadsheet per patient: this makes aggregate data analysis and reporting difficult. Once entered into the model, there is limited (if any) data sharing. The high number of patients and data demands a better and organized way to record and integrate all the information available. Of critical importance is additionally that the data is tightly coupled with this version / understanding of the definition of cachexia. Currently there are more than one definition, and if it were desired to assess the patient on a different definition/model, all the data would need to be reentered.

In large collaborative projects such as this, where many participants and groups wish to work in a virtually shared data and knowledge space, no researcher or individual has a global expertise. Different research groups are responsible for and interested in subsets of the data, processes, and results. Moreover, even in the same group there are roles and specializations. We need therefore to be able to guarantee to each actor the needed access to own and other data, while at the same time preserving the quality and the scope restrictions. In order to make the data and results available to all the scientists in the project in an adequate fashion, and later also to other scientists (in the spirit of open research and evidence-based medicine), data must be collected, stored and managed with the highest possible accuracy and care, and with a concurrent focus on sharing. That is, ideally the data should be stored on a central repository, maintained permanently, and be separated from the processes / analysis tools that are to be applied to the data.

It is clear from this setting that some information can be made public only after scientific publications have been submitted or accepted, and that there is an articulated access right management structure that governs the access and operations at the project, research group, specialty, and individual level concerning entire data sets, but also down to the single stored data element. An example of this is for instance the patient ID anonymization, that renders the concrete identity of the single patient accessible to a tiny subset of the project researchers.

In general, we talk here of conducting experiments, production and handling of blood and tissue samples, conduction of supervised and monitored physical exercise training sessions, management of radiologic data, and statistical analyses based on subsets of these raw data. Right now, the work is mostly done manually, with the guideline of textual descriptions, called Protocols. The person carrying on the work is mostly expert in one or two areas, and mostly for example not in statistical analysis. The collected data and intermediate results, for example a classification of the patient along the cachexia gravity scale, are kept locally, in inhomogeneous data sets (e.g. excel sheets) or stored in paper form (e.g. the questionnaires).

Our goal is to create, together with the project members, a web-based biomedical web application that allows the collection of the raw data with appropriate controls and constraints, supports their transformation into information units required by each participant while supporting the easy adaptation and evolution of the processes and the data that emerge. Importantly, this tool will provide the ability to adapt to the changing and evolving knowledge and needs, as well as from the growing needs once the external associated research groups and the public at large will join the use of the data and results.

5 Data Modeling in DyWA

The chosen approach to data modeling privileges the ease of modification and evolution. As explained in Sect. 3, it adopts a model driven approach and it offers the co-creation and co-evolution of data and process models. In the following, we explain briefly the specific approach we adopt (DyWA [27]) and illustrate on the same example the outcome after modeling.

5.1 The DyWA Approach to Integrated and Agile Data Modeling

A proper domain modeling is an essential part of the requirement analysis in a software project. In a classical setting, IT experts typically use graphical modeling techniques like UML class diagrams if they are software engineers, or entity relationship diagrams if they are information systems or database experts. In general, changes or extensions to the domain model underlying and software application are however not primarily foreseen. Refactorings and changes of the DB schema lead to complex, global reaching, and error-prone migrations and therefore it is possibly avoided in the classical development process. Agile methods [19] try to counter this schema lock-in syndrome at least at the level of the business logic, encouraging short and targeted development cycles as a means for a more flexible reactivity to change requests. The Dynamic Web Application (DyWA) [27,10] offers an additional agile handling also of the data structures of an application, and integrates the definition and management of the domain model into the process management of jABC4. This way, a truly collaborative design and evolution of data and processes puts for the first time the evolution of data domain and behavior on an equal footing.

In this concrete case, the processes that are currently textually or implicitly described in protocols should be collaboratively modeled, and formalized in (agile) process models. As shown in Fig. 3, the prototype based interaction of domain modeling and process modeling, in short cycles successively defines, improves, refines, and diversifies the data and the operations on and around them. The lead, ownership, and the decision power are in the hands of the researchers,

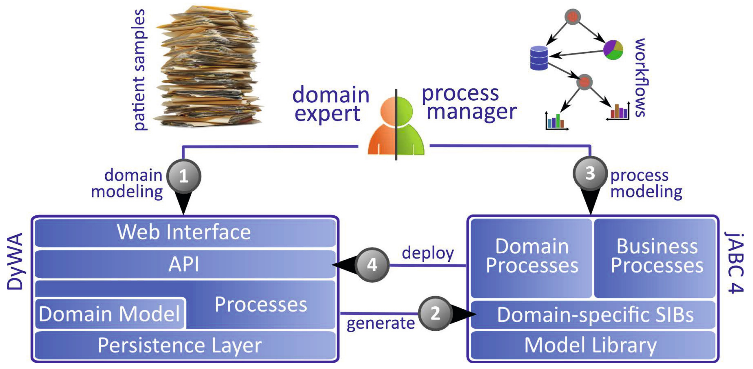


Fig. 3. Agile collaborative software development with DyWA and jABC4

in a competence based style [5]. Data and process models should be dynamically modifiable, with an ease of modification that closely follows the change of needs, the growth of the groups and the diversification of the actors.

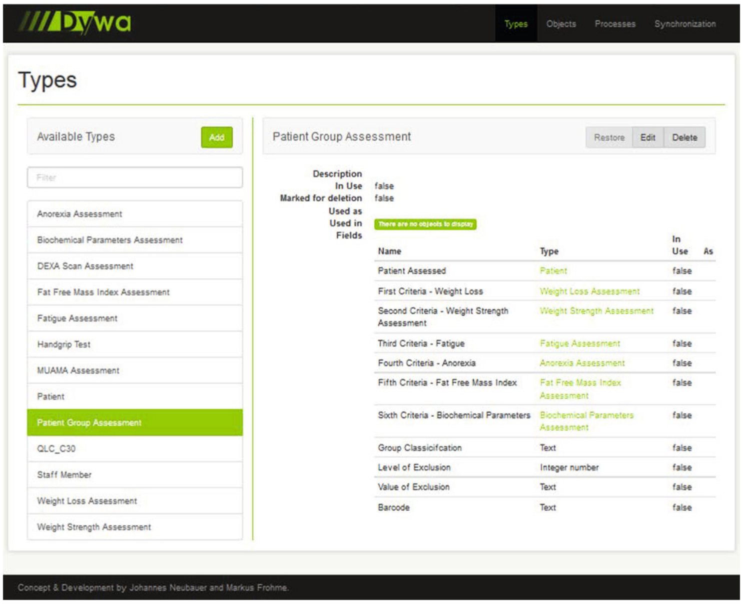
The concrete interplay of the two tools supports the four phases:

1. Domain modeling
2. Automatic Generation of the domain-specific data types and of the code for the CRUD operations upon them, with deployment in the jABC4 as a domain specific collection of services
3. Process definition and validation/verification in jABC, using these domain specific services plus a growing collection of similar services that are either domain-independent (e.g. export as file, or as CSV) or provene from other domains. Examples of such domains are geo-information and visualization, e.g. to create maps of prevalence of certain phenomena, or statistical analysis with Matlab or GNU R.
4. Code generation for the finished and verified processes and their deployment into DyWA, that now exploits its nature as web application to immediately make these processes available to the users worldwide.

This way, we achieve the central repository and easy access to it that we had set as primary criterion for the collection, management, and fruition of the data. Details on the characteristics of jABC that make it particularly adequate as a high-assurance process design tool for scientific workflows in a volatile environment (like in the design of scientific experiments) are available in [26,25] and [12,14].

5.2 How the Example Has Been Treated

The modeling in DyWa was carried out by three senior students of Business Information Systems in the business school in CalPoly. They had some explanations on the overall project structure by other team members, and a brief



Types

Available Types

- Anorexia Assessment
- Biochemical Parameters Assessment
- DEXA Scan Assessment
- Fat Free Mass Index Assessment
- Fatigue Assessment
- Handgrip Test
- MUAMA Assessment
- Patient
- Patient Group Assessment
- QLC_C30
- Staff Member
- Weight Loss Assessment
- Weight Strength Assessment

Patient Group Assessment

Description	
In Use	false
Marked for deletion	false
Used as	
Used in	There are no objects to display
Fields	

Name	Type	In Use	As
Patient Assessed	Patient	false	
First Criteria - Weight Loss	Weight Loss Assessment	false	
Second Criteria - Weight Strength Assessment	Weight Strength Assessment	false	
Third Criteria - Fatigue	Fatigue Assessment	false	
Fourth Criteria - Anorexia	Anorexia Assessment	false	
Fifth Criteria - Fat Free Mass Index	Fat Free Mass Index Assessment	false	
Sixth Criteria - Biochemical Parameters	Biochemical Parameters Assessment	false	
Group Classicifcation	Text	false	
Level of Exclusion	Integer number	false	
Value of Exclusion	Text	false	
Barcode	Text	false	

Fig. 4. The Patient Group Assessment type in DyWA. Type collection (on left) and complex type type composition (on right). The green types are domain-specific types modeled after the spreadsheet.

introduction to DyWA and to jABC4. They had previous experience of domain modeling in the traditional E/R style, but this was not a prerequisite. They were provided with the questionnaire and the spreadsheet of Section 4, they interpreted the spreadsheet and turned its data definition part into a collection of types in DyWA.

As we see in Fig. 4, there is now in DyWA a collection of types (on the left) corresponding to the sub-elements of the excel spreadsheet. They include the basic demographics and anthropometrics (in the type Patient), but also types for other more complex measurements like the MUAMA Assessment, that includes the computation of a MUAMA Score, and is used in the Fat Free Mass Index Assessment. The basic data items, like age, gender, height, weight, are collected as primitive data, in accordance with the data collection principles enunciated in Sect. 3.1. Derived values like the BMI or the MUAMA Score, are going to be computed by means of little computational processes, and the dependencies between types, in terms e.g. of the Used-in relation, are automatically tracked by the DyWA infrastructure.

In terms of agility and long term evolution support of data structures, DyWA features a built-in staged development that makes change management easier. Changes in the definition of data types are not immediate (like in excel, which is an interpreted programming environment), but decoupled from the running version. Therefore any change is first only marked for change or for deletion, and there are built-in checks that make sure that no change will become operational if the processes that use that data have not been correspondingly updated. Users can therefore safely experiment on the data and process definitions, without affecting the operational version that continues to run undisturbed.

In terms of processes, for the moment we are addressing the built-in computations of the excel classification spreadsheet and modeling them as jABC4 processes that use the CRUD operations automatically generated by DyWA for all the types there defined. The idea is to create a library of domain-specific SIBs (or units of operation) and processes that users can then easily modify and adapt, in a *domain-bootstrapping* fashion.

6 Discussion and Perspectives

The role of long term data management usability coupled with tools to provide security, access, privacy and quality is paramount in the realm of healthcare systems. In order to react positively to changes in processes, tools, methods, and data, the systems of tomorrow must embrace agility. Static designs built with old school thinking need to be replaced with tools that provide the aforementioned performance gains. Data design and collection systems such as DyWA are the future in data management and especially in data management of mission critical applications where potential failures can negatively impact research results and the concomitant informing of health care policies, practice and research.

Importantly, research results can no longer be siloed. Transparency, appropriate shared access, and knowledge growth are key to the health of today's international society. Data management systems that adopt such a perspective with realize the high potential gains that data analytics can bring to the provision of new understandings and ultimately treatments and hopefully cures.

As noted earlier, an important performance factor in medical research systems is the security and tracking of all data transactions, e.g. a sequence of CRUD operations. Importantly, identifying and maintaining role based access techniques at entry to facilitate and restrict access to the correct individuals must be embedded directly into the design of the meta data of the system as well as into the technical infrastructure and enables this functionality.

Our work with the ‘engaged professional’ suggests that the power in our thinking and approach enables these end users to be proactive in the creation and use of the data management systems and repositories. Deskilling these technologies provides an impact that is immeasurable in the creation of new medical knowledge through domain skilled design and through the building of international research alliances through data collaborations.

Acknowledgments. We thank Markus Frohme and Dennis Kühn (TU Dortmund) for their support with the modeling tools (DyWA and jABC), Aaron James, Nathan Chen and Jessica Liang (CalPoly), Michele Joana Alves, Katrin Radloff and Emidio Matos (USP) for their help in understanding the project organization. This work was partially funded with the following grants: PROBRAL 369/12 (2011-2013), ITSy EU FP7-ICT-2009-5 Project 258058 (2010-11), FAPESP 12/50079-0.

References

1. Epirare, the European platform for rare disease registries, `http://www.epirare.eu/`
2. Evidence-based medicine, `http://ebm.bmj.com/site/about/whyread.xhtml`
3. Global rare diseases patient registry and data repository, `http://rarediseases.info.nih.gov/research/pages/43/global-rare-disease-patient-registry-and-data-repository`
4. Argilés, J.M., Busquets, S., López-Soriano, F.J.: Anti-inflammatory therapies in cancer cachexia. European Journal of Pharmacology 668(suppl. 1), S81–S86 (2011), `http://www.sciencedirect.com/science/article/pii/S0014299911007783`, pharma-Nutrition
5. Rasche, C., Margaria, T., von Reinersdorff, A.B.: Value delivery through it-based healthcare architectures: towards a competence-based view of services. 25 Jahre ressourcen- und kompetenzorientierte Forschung: der kompetenzbasierte Ansatz auf dem Weg zum Schlüsselparadigma in der Managementforschung, pp. 417–443 (2010)
6. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. The MIT Press, Cambridge (1999)
7. European Organisation for Research and Treatment of Cancer (EORTC): Qlq-c30: quality of life questionnaire, `http://groups.eortc.be/qol/eortc-qlq-c30`
8. Evans, W.J., Morley, J.E., Argilés, J., Bales, C., Baracos, V., Guttridge, D., Jatoi, A., Kalantar-Zadeh, K., Lochs, H., Mantovani, G., Marks, D., Mitch, W.E., Muscaritoli, M., Najand, A., Ponikowski, P., Rossi Fanelli, F., Schambelan, M., Schols, A., Schuster, M., Thomas, D., Wolfe, R., Anker, S.D.: Cachexia: a new definition. Clin. Nutr. 27(6), 793–799 (2008)
9. Fearon, K., Strasser, F., Anker, S.D., Bosaeus, I., Bruera, E., Fainsinger, R.L., Jatoi, A., Loprinzi, C., MacDonald, N., Mantovani, G., Davis, M., Muscaritoli, M., Ottery, F., Radbruch, L., Ravasco, P., Walsh, D., Wilcock, A., Kaasa, S., Baracos, V.E.: Definition and classification of cancer cachexia: an international consensus. The Lancet Oncology 12(5), 489–495 (2011)
10. Frohme, M.: Agile Domänenmodellierung für prozessgesteuerte Webanwendungen. Bachelor thesis, TU Dortmund (2013)
11. Jonsson, B., Margaria, T., Naeser, G., Nyström, J., Steffen, B.: Incremental requirement specification for evolving systems. Nordic J. of Computing 8, 65–87 (2001), `http://dl.acm.org/citation.cfm?id=774194.774199`
12. Jörges, S., Lamprecht, A.L., Margaria, T., Schaefer, I., Steffen, B.: A Constraint-based Variability Modeling Framework. International Journal on Software Tools for Technology Transfer (STTT) 14(5), 511–530 (2012), `http://www.springerlink.com/content/e453185h77261371/`
13. Jörges, S., Margaria, T., Steffen, B.: Genesys: service-oriented construction of property conform code generators. Innovations in Systems and Software Engineering 4(4), 361–384 (2008)
14. Lamprecht, A.-L.: User-Level Workflow Design. LNCS, vol. 8311. Springer, Heidelberg (2013)

15. Lamprecht, A.-L., Margaria, T. (eds.): Process Design for Natural Scientists - An Agile Model-Driven Approach. CCIS, vol. 500. Springer, Heidelberg (2014)
16. Lamprecht, A.-L., Margaria, T., Steffen, B.: Seven Variations of an Alignment Workflow - An Illustration of Agile Process Design and Management in Bio-jETI. In: Măndoiu, I., Wang, S.-L., Zelikovsky, A. (eds.) ISBRA 2008. LNCS (LNBI), vol. 4983, pp. 445–456. Springer, Heidelberg (2008), http://dx.doi.org/10.1007/978-3-540-79450-9_42
17. Lamprecht, A.L., Naujokat, S., Margaria, T., Steffen, B.: Semantics-based composition of EMBOSS services. Journal of Biomedical Semantics 2(suppl. 1), S5 (2011), http://www.jbiomedsem.com/content/2/S1/S5
18. Margaria, T., Kubczak, C., Njoku, M., Steffen, B.: Model-based Design of Distributed Collaborative Bioinformatics Processes in the jABC. In: Proceedings of the 11th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 2006), pp. 169–176. IEEE Computer Society, Los Alamitos (2006)
19. Margaria, T., Steffen, B.: Agile IT: Thinking in User-Centric Models. In: Margaria, T., Steffen, B. (eds.) ISoLA 2008. CCIS, vol. 17, pp. 490–502. Springer, Heidelberg (2008)
20. Margaria, T., Steffen, B.: Continuous Model-Driven Engineering. IEEE Computer 42(10), 106–109 (2009)
21. Margaria, T., Steffen, B.: Service-Orientation: Conquering Complexity with XMDD. In: Hinchey, M., Coyle, L. (eds.) Conquering Complexity, pp. 217–236. Springer, London (2012), http://dx.doi.org/10.1007/978-1-4471-2297-5_10
22. Müller-Olm, M., Schmidt, D., Steffen, B.: Model-Checking - A Tutorial Introduction. In: Cortesi, A., Filé, G. (eds.) SAS 1999. LNCS, vol. 1694, pp. 330–354. Springer, Heidelberg (1999), http://dx.doi.org/10.1007/3-540-48294-6_22
23. Neubauer, J.: Higher-Order Process Engineering. Phd thesis, Technische Universität Dortmund (2014), http://hdl.handle.net/2003/33479
24. Neubauer, J.: Higher-Order Process Engineering: The Technical Background. Tech. rep., Technische Universität Dortmund (April 2014), http://hdl.handle.net/2003/33102
25. Neubauer, J., Steffen, B.: Plug-and-Play Higher-Order Process Integration. IEEE Computer 46(11), 56–62 (2013)
26. Neubauer, J., Steffen, B.: Second-Order Servification. In: Herzwurm, G., Margaria, T. (eds.) ICSOB 2013. LNBIP, vol. 150, pp. 13–25. Springer, Heidelberg (2013), http://dx.doi.org/10.1007/978-3-642-39336-5_2
27. Neubauer, J., Frohme, M., Steffen, B., Margaria, T.: Prototype-Driven Development of Web Applications with DyWA. In: Margaria, T., Steffen, B. (eds.) ISoLA 2014, Part I. LNCS, vol. 8802, pp. 56–72. Springer, Heidelberg (2014)
28. Steffen, B., Margaria, T., Nagel, R., Jörges, S., Kubczak, C.: Model-Driven Development with the jABC. In: Bin, E., Ziv, A., Ur, S. (eds.) HVC 2006. LNCS, vol. 4383, pp. 92–108. Springer, Heidelberg (2007), http://dx.doi.org/10.1007/978-3-540-70889-6_7

Domain-Specific Business Modeling with the Business Model Developer

Steve Boßelmann and Tiziana Margaria

Department of Computer Science
University of Potsdam
August-Bebel-Str. 89
14482 Potsdam, Germany
{bossel,margaria}@cs.uni-potsdam.de

Abstract. Discussing business models of companies and organizations based on graphical representations that emphasize essential factors has increased in recent years, particularly from a business perspective. However, feasible implementations of business modeling tools are rare, as they tend to be domain specific but at the same time tailored towards the requirements of a heterogeneous group of stakeholders. We present the Business Model Developer (BMD) and the underlying framework for the creation of domain-specific business models. The approach relies on the definition of a domain-specific library of model components as well as structured parameters with different scopes, i.e. declared areas of applicability. This setup forms the basis for the development of custom techniques for model analysis. We align the development process with the approach of Extreme Model Driven Development (XMDD) to keep it as simple as possible for domain experts to contribute in tailoring the tool towards specific needs. Furthermore, we present a practical application in the healthcare domain, as the BMD has been developed and applied in the course of a joint project in the area of Personalized Medicine.

Keywords: business model, personalized medicine, model-driven software development, simplicity.

1 Introduction

Modeling tools like the *Business Model Canvas* (BMC) [1] receive significant attention in the area of business strategy. However, the term ‘tool’ is ambiguous and in this case should be interpreted as a ‘best practice’ approach to gathering the most important business-related aspects by asking the right questions. However, from a computer science perspective there are virtually no sophisticated tools that support business model design by means of applying formal semantics, neither in general nor for specific areas of application. This lack of formalism reflects the fact that on the one hand there is neither common understanding about a suitable ontology nor about component types that business models should actually comprise. Hence particular model characteristics depend on the application area as well as on the actual business domain.

As a consequence, the tool development process depends on inter-disciplinary collaboration and communication. It requires immediate contribution of a heterogeneous group of stakeholders involving business management, finance and economics researchers, depending on the tool's field of application and intended purpose. Business models – like models in general – represent a common language for these stakeholders with often entirely different disciplinary backgrounds, as they enable the depiction of key aspects to support discussions and decision-making [2]. However, most of these stakeholders typically lack broad knowledge of formal models and software engineering skills. Hence, a decent amount of simplicity throughout the development lifecycle is key for success. The notion of simplicity as a driving paradigm in information system development has been explicitly identified as an important research topic, yet it is still poorly understood [3,4] and not widely adopted in research [5].

In order to integrate stakeholders with different disciplinary backgrounds in the modeling and comprehension process in a possibly simple and intuitive way, we follow the approach of *Extreme Model Driven Development* (XMDD) [6], an advancement of *Continuous Model-Driven Engineering* [7]. In this context, we direct our development efforts towards leveraging the jABC framework [8] as our main development environment as it supports XMDD in a consistent manner.

Regarding the structure of this article, first of all we will present the basic modeling concepts of the Business Model Developer in Section 2, especially the domain-specific setup based on a component library and structured parameters. In Section 3 we describe the practical setting of the healthcare-related joint project where the tool has been developed and applied in practice. Here we also describe the main features of the tool's user interface, the actual steps for designing business models as well as an example of applied model analysis. Finally, Section 4 concludes with some thoughts and ideas for future work.

2 Domain Specification

The presented approach to business modeling is based on the *Business Model Ontology* (BMO) [9] that results from Osterwalder's research comprising balanced score cards [10], value chains [11], and stakeholder analysis [12]. The BMO defines nine separate model segments that represent semantic categories. Reasonably arranged, they form the well-known partition of the Business Model Canvas. We will refer to these as 'canvas categories' throughout this article. The BMC is a conceptual template to be filled with the actual modeling entities to depict "*a description of the value a company offers to one or several segments of customers and the architecture of the firm and its network of partners for creating, marketing and delivering this value and relationship capital, in order to generate profitable and sustainable revenue streams*" [9]. We will refer to these modeling entities as 'model components' whenever we want to stress that they constitute the actual model. On the other hand, each of these model components represents a business-related entity to be referred to as 'business item' throughout this article.

With the definition of the canvas categories the Business Model Canvas comprises a description of nine specific containers to hold items but only loosely restricts the kind of items to be placed inside. The Business Model Canvas is a handcrafted approach to business model design. Typically, in workshops and moderated modeling sessions users brainstorm, create notepads with keywords and place them on a physical Business Model Canvas such as on a flipchart. Starting with a bare canvas it remains up to the modeler to create a design that is semantically correct. This holds for the pen-and-paper approach as well as for the commercial BMC software tool [13], which basically allows for the placement of generic, virtual notepads onto a virtual canvas.

In contrast, our approach facilitates a pre-defined domain-specific setup of the modeling environment. The definition of this setup takes place in a distinct customization step preceding the actual model design. Typically, different stakeholders are involved in this customization step, spanning application experts, domain experts as well as the modeler representing the actual user from the application domain the Business Model Developer is tailored to. This way, user needs can be communicated and considered in an immediate manner. The setup is not fixed but can at any time be adapted to changes in market conditions and stakeholder requirements. This allows for an iterative development and continuous improvement of the tool's setup to best reflect the application domain.

A domain-specific setup of the Business Model Developer consists of the definition of a library of building blocks that represent available model components, along with a list of parameter definitions to be applied to these model components. The following sub-sections provide a more detailed description of these aspects.

2.1 Library of Building Blocks

The core of the proposed domain-specific setup of the Business Model Developer is the definition of a library of building blocks that represent available model components. Providing such a library of building blocks has two advantages over approaches that merely rely on generic elements, like labeled notepads.

- On the one hand, listing appropriate model components offers the modeler an incentive to discover unused potential as they represent business items that are relevant for the considered domain and hence outline aspects that otherwise might not have been taken into account.
- On the other hand, a library of building blocks provides the components for a sound business model instead of serving the user a completely blank canvas with which to begin.

As this way the modeling part as such does not start with a bare canvas, the building blocks provide valuable assistance and guidance e.g. to entrepreneurs for designing first business models. This approach of providing building blocks in such a fashion is similar in concept to graphics tools that provide the feature of drawing basic shapes instead of leaving the user with a freehand pen and some good advice on how to do it. Additionally, a pre-defined set of building blocks serves as a shared vocabulary that participants with different professional background can agree upon in order to increase communicability and information exchange amongst them. This understanding is essential because business models – just like models in general – are created to point out crucial aspects and hence support decision-making.

The underlying framework of the Business Model Developer provides full interchangeability regarding the library of building blocks. This way the model components can be specified for the actual application domain and tailored towards the modelers' requirements. Furthermore, each building block is determined for a specific canvas category. Hence the whole library can be sorted according to the canvas categories. During the modeling process this enables filtering of applicable model components to exactly those that can be placed inside a respective canvas category. This way, the modeling software supports the modeler in the creation of models that are syntactically correct by preventing misplaced model components. This is a real advantage over the unrestricted placement of generic labeled notepads.

2.2 Classification of Business Items

In general, the elements within our proposed library of building blocks do not correspond to individual business items but represent classes of items. As an example, the library would contain an abstract element '*Research Institution*' for the canvas category '*Key Partners*' instead of listing all universities that exist. Hence, in order to express that '*University of Potsdam*' is a key partner, the modeler can select the building block '*Research Institution*' which triggers the creation of a corresponding model component. This model component is interpreted as an instance of the class represented by the selected building block. Thus, the creation process is referred to as 'instantiation' and the class represented by the building block is referred to as the model component's type. A single canvas category can hold multiple instances of a single class, i.e. one single building block. The modeler can specify a custom name for each instance according to the respective business item it is intended to represent. To conclude our illustrative example, the modeler would specify the name 'University of Potsdam' for the instantiated model component of class 'Research Institution'.

The classes of business items represented by building blocks can be defined in such a manner that they correspond to very specific sets of items. This way, a building block can indirectly even represent a single business item. As the type of each model component remains accessible over the model's lifetime, this information can be evaluated by software, which in a succeeding step might trigger some reasoning based on this information. In particular, type information is an essential aspect for the definition of rules to be checked and verified at model design time as well as the application of any model analysis technique. Hence, the structure of the library of building blocks not only is tailored to the respective business domain in terms of providing a common language. It also needs to comply with analysis requirements eventually formulated by stakeholders interested in evaluation. For example, if a specific analysis technique requires the distinction between different types of research institutes within the partner network, the library of building blocks can be defined accordingly by providing separate building blocks for each of these types.

2.3 Specification of Building Blocks

The classification of business items to be used in the design of business models facilitates the restriction of items to be placed inside a specific canvas category. The required knowledge base enabling this evaluation can be specified by an appropriate ontology model, as ontologies are used to formally describe the terms of a domain as well as the relations between them [14]. Generally, the vocabulary of ontology models bases on the concept of classes and individuals, the latter representing instances of classes. In order to avoid confusion, they are referred to as 'ontology classes'.

Ontology Models. In the context of the Business Model Developer we use ontology models to specify a taxonomy of building blocks. The hierarchical structure of this taxonomy corresponds to the set of canvas categories, i.e. it contains a list of building blocks for each of the latter. Such a list can be specified by naming the identifier of an ontology model as well as a set of ontology classes within this model. The actual instances of these ontology classes are considered as building blocks for the respective canvas category. These instances are specified as individuals within the same ontology model. These individuals are connected to the respective ontology class via an *instanceOf* relation. We refer to the combination of ontology identifier and related ontology classes as 'ontology sector'.

As the desired set of building blocks for a specific canvas category can be spread over multiple ontology models, the Business Model Developer allows for the specification of multiple ontology sectors for each canvas category. Taking inheritance into account, the actual list of building blocks for a specific canvas category is constructed by collecting and combining the instances of the ontology classes for each respective ontology sector.

The presented approach for the specification of building blocks allows for using existing ontology models. For many domains, ontologies have been maintained that are freely accessible via online libraries[1]. Alternatively, custom ontology models might be defined and tailored towards specific use cases. We are using the Web Ontology Language (OWL) [15] for this approach. In particular, we use the *OntED* plugin of the jABC framework [8] that provides a graphical editor for the creation of OWL-based ontology models. Hence, according to the XMDD approach [5] we create a graphical model for specific needs within our main development environment.

Custom Taxonomies. The specification of a custom taxonomy of building blocks is straightforward. The corresponding ontology model contains ontology classes that are classified according to the canvas categories, i.e. there is an OWL class for each canvas category, respectively. Subsequently, any building block that should be assigned to this category is defined as an instance of this class by creating an OWL individual connected via the *instanceOf* relation. As an example, Fig. 1 shows a tiny snippet of the ontology model created this way. It shows that according to the canvas

[1] Ontology data sharing, e.g. `http://www.ontobee.org` and `http://www.obofoundry.org`

category '*Value Propositions*' the ontology contains the OWL class *ValueProposition.* Building blocks for this canvas category are instances of this class, represented as OWL individuals.

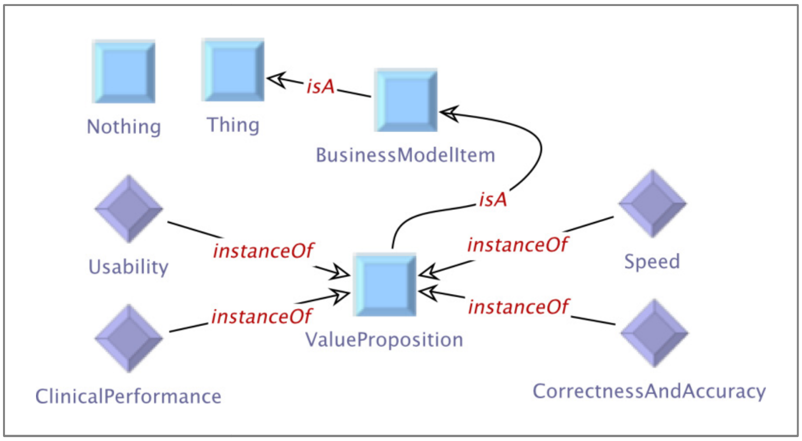


Fig. 1. Snippet of the ontology model

The specification of building blocks with the use of ontology models is evaluated at runtime. Hence, the Business Model Developer immediately reacts on changes to the ontology model. Based on the information within the ontology, the building blocks are filtered for each canvas category. The restricted use of model components achieved this way allows for avoiding mistakes during the design process as the correct placement on the canvas is enforced.

Finally, a well-defined set of classified building blocks paves the way for the analysis of a business model as well as specific model checking routines that would not be applicable on non-classified entities. However, in order to not restrict the creative freedom of the modeler the Business Model Developer allows for the ad-hoc extension of the component library by custom-typed model components whenever the user needs to create a business item that does not fit in the pre-defined type scheme represented by the building blocks. At design time, model components of custom type can be used just like any other model component of pre-defined type, but they will not be considered for model analysis.

2.4 Parameterization of Business Items

Besides this library of building blocks the domain-specific setup of the Business Model Developer comprises a list of parameter definitions to be applied to model components. As these components represent business items, parameters allow for a more detailed characterization of a business model by the enrichment of its components with user-provided content in the form of parameter values. To name some examples, these parameters enable the user to state the type of interaction with a key partner, the costs of a specific key resource or the value of a revenue stream.

Parameter Scopes. Parameters can be generic, i.e. they are existent at every model component but they can also be specific for a limited subset. Especially the subset of building blocks for a specific canvas category is of particular interest in this context. Hence we use the concept of scopes for parameters. In particular, right in between generic parameters (canvas scope) and component-specific parameters (component scope) we introduce category-specific parameters (category scope). Besides the straightforward definition of component-specific parameters, the technical realization is based on the definition of parameter templates to be linked with canvas categories or the canvas itself. At instantiation time of a specific model component the parameter templates of the respective category as well as the ones of the canvas are instantiated as well and linked to this component. From now on they are treated just like any normal parameter that is directly associated with a model component.

Parameters with limited scope override parameters with wider scope if the respective parameter keys are identical. This makes it possible to define parameters for all the items of a specific canvas category without knowing the actual component library but at the same time allow amending some of these parameters for a single component wherever it is required. However, a parameter's scope is not visible to the user of the Business Model Developer because from a user's perspective scopes are considered technical detail irrelevant for the actual use of parameters. The use of a parameter as well as its appearance for the user is realized independently of its actual scope.

Category-Specific Parameters. An example of a category-specific parameter is the parameter '*Timing of Interaction*'. With the help of this parameter the user can provide details on when exactly the interaction with a specific associate partner takes place. Hence, the scope of this parameter is the single canvas category '*Key Partners*'. The parameter is applicable for any model component to be instantiated within this canvas category, independent of the actual component type. This illustrates that category-specific parameters can be defined independently of the actual set of model components related to this category. Parameters can even be domain-independent. In our illustrative example, the parameter '*Timing of interaction*' is applicable to whatever model component is instantiated within the category '*Key Partners*', as it only exploits the fact that this component is meant to represent an associated partner in the context of a business model and there is some kind of interaction with this partner at some point in time.

In some cases, the scope of a parameter can span multiple canvas categories. Cost factors are a typical example. They can be specified for most of the model components, while for some categories (e.g. customer segments) specifying costs makes less sense. Hence, they are not defined as generic parameters with global scope. On the other hand, for example any key partner, key resource or key activity can induce costs. Hence, the scope of the parameter '*Cost Factor*' is defined as a set of canvas categories.

Parameter Structure. From a technical perspective, the Business Model Developer provides a default parameter construct that covers literal value types as well as custom value objects. It is designed to either hold a single value or a list of values. In both cases a set of selectable values can be specified from which the parameter value (or multiple values) can be chosen from. This parameter structure with a pre-defined list of selectable values is similar to the structure of multiple-choice questionnaires. Hence it enables the transfer of survey structures in an immediate manner. At the same time, it allows for an evaluation, just like the types of model components can be considered e.g. in the context of any analysis algorithm. User-specified parameter values remain accessible over the complete lifetime of the business model. Depending on the combination of parameter values various characteristics might be identified and depending on the latter appropriate rules might be triggered or conclusions might be drawn. In combination with the types of the respective model components that hold the parameters this opens a lot of potential for analysis techniques that would not be applicable in tools that solely rely on pure graphical representation.

The depicted approach bases on a generic parameter structure and already covers a wide range of common parameter constructs. However, the underlying framework of the Business Model Developer does not restrict the type of parameters and allows for their extension by programming custom parameters into code.

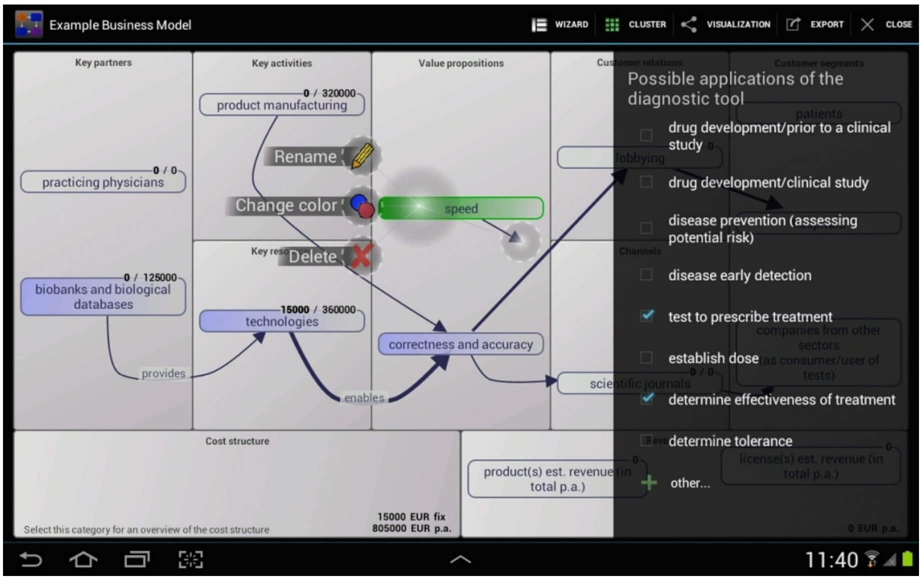
3 Business Modeling in Personalized Medicine

The Business Model Developer has been developed in the course of the joint project *Service Potentials in Personalized Medicine* (DPM) [16], funded by the German *Federal Ministry of Education and Research* (BMBF). The main objective of the project was a market analysis of Personalized Medicine to identify key actors, drivers and barriers, that included the analysis of current and future business models within this specific market segment. The Business Model Developer was developed not only under consideration of the project's findings but also practically applied in interview sessions with industry experts as well as repeatedly tested and evaluated by project partners. Experiences and insights achieved this way have directly influenced further improvement of the Business Model Developer.

The frontend of the Business Model Developer that has been developed for the DPM project is an Android-based App for Tablets. The following subsections comprise a short description of the tool's user interface as well as how to create business models with it.

3.1 Filling the Canvas

The visualization of the workspace is based on the design of the Business Model Canvas and adopts the arrangement of its nine canvas categories. The modeler successively enriches them with model components from the library of building blocks. This library is accessed via the context menu of a canvas category and the listed building blocks are restricted to those that are suitable for the respective category. This way, the user interface avoids the misplacement of model components in a rigorous manner.



Possible applications of the diagnostic tool

- ☐ drug development/prior to a clinical study
- ☐ drug development/clinical study
- ☐ disease prevention (assessing potential risk)
- ☐ disease early detection
- ✓ test to prescribe treatment
- ☐ establish dose
- ✓ determine effectiveness of treatment
- ☐ determine tolerance
- + other...

Fig. 2. Screenshot of the canvas with colored model components and labeled relations, an item's context menu (middle) and the overlay inspector showing the item's parameter values (right).

Domain-Specific Components. In the context of the DPM project the library contains components that are most likely for business models in the area of Personalized Medicine. The building blocks have been identified by the evaluation of surveys and interviews with industry experts with a focus on diagnostic companies [17]. As an example, building blocks for the category *Key Partners* span general entities like *Research Institutes*, *Companies* and *Investors*. At the same time the Business Model Developer lists items that are very specific for the domain of diagnostic companies, such as *Biobanks*, *Biological Databases* as well as *Researching Physicians*. Analogously, the building blocks for other canvas categories are tailored towards this distinct field of application.

Selecting one of the listed building blocks leads to a dialog that asks the user for the name of the model component to be instantiated. Furthermore, associated parameters are listed for the user to provide appropriate values. Depending on the type of parameter, the user can select from pre-defined values or provides custom textual or integer input. Having done so, the new model component is instantiated according to the user's input and visualized by means of a labeled rectangular shape inside the dedicated area of the canvas representing the respective canvas category. The user is able to assign custom colors to the items' shapes in order to emphasize special group memberships of items. Fig. 2 shows a screenshot of an example model as well as the main components of the user interface.

Accessing Parameters. Parameters and their values are not visualized on the canvas directly. Instead, an overlay dialog (referred to as an inspector) is shown that covers parts of the canvas whenever a model component is selected. The inspector lists detailed information on the component's type as well as all of the corresponding parameters and respective parameter values. The latter may be manipulated in the inspector directly, be it simple key-value pairs or structured parameters with predefined value lists. As soon as the item is unselected any changes are applied and the inspector fades out.

The parameter '*Cost Factor*' represents an exception from the inspector-based approach of parameter visualization. These parameters are displayed in the upper corner of the respective item's rectangle shape. Both values, fix costs and costs per anno are displayed at the same time. This provides a fast overview over the model's different cost factors while maintaining the visual link to the associated model component. Additionally, the computed total costs are depicted in the lower corner of the canvas category '*Cost Structure*', again separated into fix costs and costs per anno. As an alternative, the inspector shows a list of all cost factors whenever the canvas category '*Cost Structure*' is selected. They appear like parameters grouped by the name of the model component that holds the respective parameter. Their values can be changed directly via the inspector. Doing so will update the value of the costs parameter of the respective model component.

3.2 Modeling the Relations

The Business Model Developer provides an additional feature for emphasizing interrelations between business items, like for example dependencies. The modeler can specify relations between two particular model components. As business models typically contain various types of relations between their components, facilitating the explicit specification of these relations provides genuine added value for model creation. From an opposite perspective, without the ability of specifying relations business models would lack essential information needed for its understanding and interpretation. As an example, providing lists of key resources and key activities does not state anything about how resources are allocated to the respective activity. However, resource allocation can be understood as a type of relation between resources and activities. Hence the allocation can be clearly specified by linking each resource to the respective activity it is related to. This and other common types of relations within business models have been identified and described along with the creation of the Business Model Ontology [9]. However, the Business Model Canvas does not cover the explicit creation of relations. Hence, the modeling of relations by means of the Business Model Developer represents another valuable extension of the underlying BMC framework.

The graphical representation of relations is based on directed edges that may be labeled. While the direction of the edge supports the interpretation of the relation, labeling the edge allows to clearly specifying the type of the represented relation between the respective business items. That means, in order to specify a relation between two specific business items the modeler creates a directed edge between the model components that represent these items and labels it accordingly. Fig. 3 shows some illustrative examples for different types of relations.

The visualization of edges can be customized by means of variable line thickness. This feature can be used to emphasize different weighting of relations in a qualitative manner. In particular, thicker lines stand for stronger relations. Fig. 3 shows an illustrative example of an activity that is supported by two different key partners. The thickness of the edges representing the relation 'supports' differs, suggesting the interpretation that one of the actors is more supportive than the other.

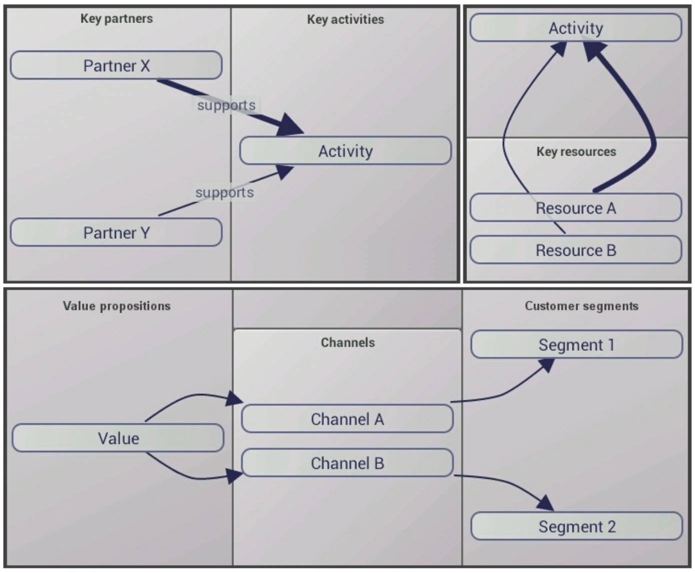


Fig. 3. Examples of the use of relations. Top-left: Using line thickness to emphasize qualitative differences between two relations of the same type. Top-right: Resource allocation. Bottom: Indicating different channels for different customer segments.

The user can further adjust the visualization of edges by means of deviation points. They are interpreted as reference points that mark a path the respective edge is drawn along. This feature is solely a graphical aspect, as deviation points do not have any semantics.

3.3 Guided Modeling with the 'Wizard'

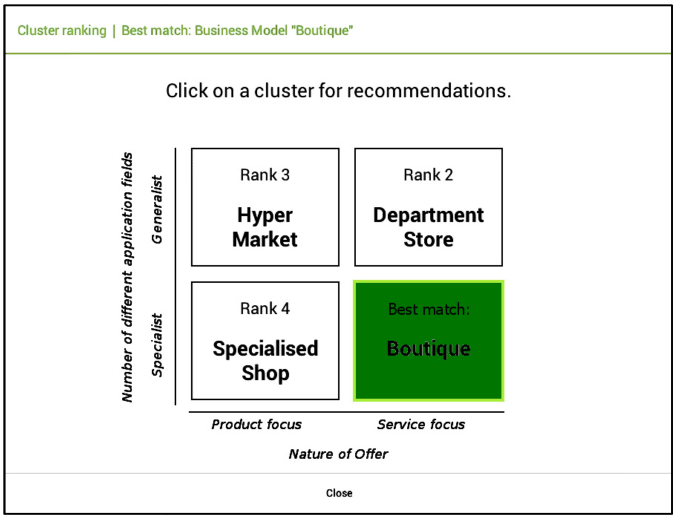
While the above concepts of creating business models rely on direct interaction with the canvas, the Business Model Developer provides an alternative way to develop a model. In this approach the user accesses a 'Wizard' which is dialog-based, i.e. the modeler follows a structured approach similar to a questionnaire. The Wizard lists the nine categories of the canvas in a pre-defined order and guides the user through the necessary steps to fill them with model components. This guidance is realized by means of significant descriptions and supporting hints regarding the meaning of the different categories as well as the purpose of parameters.

This Wizard-based approach is especially helpful for beginners as it supports the creation of first business models in a directed fashion, improving completeness.

The actual graphical representation of the business model is successively created in the background based on the information the modeler provides during the interaction with the Wizard. The user can trigger the creation of this canvas-based view at any time, as filling some categories might as well be skipped. However, the Wizard does not only provide initial guidance. It is intended to be used as an alternate view on the business model. Thus, the Wizard can be re-entered at any time and for any model state.

3.4 Business Model Analysis

Typed model components as well as structured parameters with limited value ranges form the basis for the application of model analysis. In the course of the DPM project a systematic analysis technique has been developed by the colleagues that enables a comparison of business models [17]. This analysis is based on the collected data regarding business models of diagnostic companies, accessing the same data that has been defined via the DPM-specific library of building blocks. Based on the evaluation of this data, they identified distinct clusters that represent a partition over different instances of business models. The Business Model Developer enables the automated calculation of cluster membership and thus enables the comparison of the business model designed to current market reality in the area of Personalized Medicine. The cluster analysis makes use of the accessible information regarding the types of model components that have been used within each canvas category as well as the selected values of the corresponding parameters.


Cluster ranking | Best match: Business Model "Boutique"

Click on a cluster for recommendations.

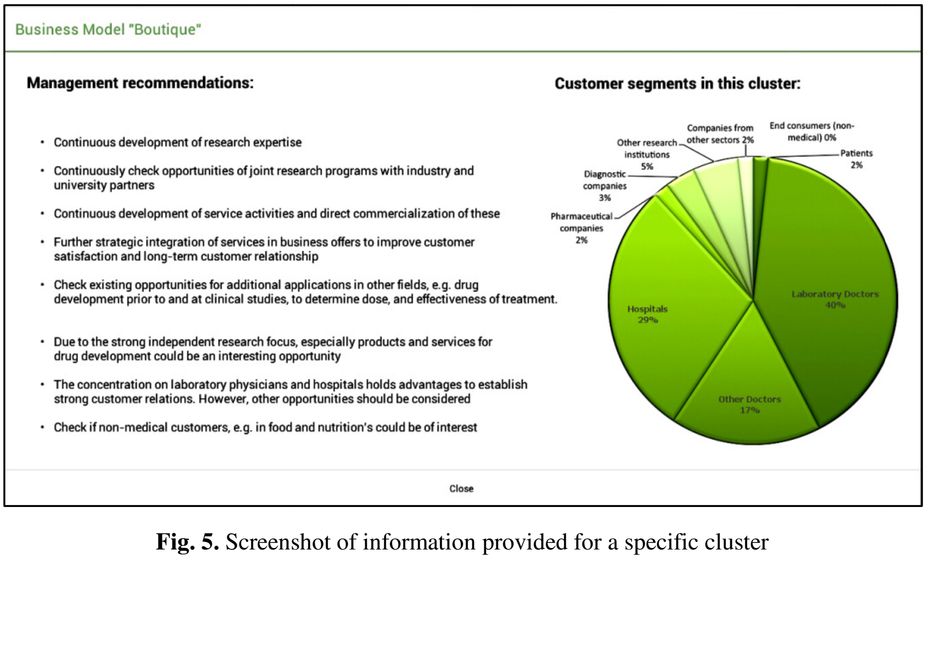
Fig. 4. Screenshot of the result of cluster analysis

In the specific context of the DPM project, focusing on diagnostic companies in the area of Personalized Medicine, the cluster analysis is based on two main dimensions. On the one hand, the type of the respective offer is analyzed. Service-centered business models are rated as completely different to those that are centered on product manufacturing. This type of offer corresponds to the type of the components placed in the canvas category ‘Key Activities’. The second dimension of the cluster analysis is the number of different application fields of the offered diagnostic product or service. This information is specified by means of distinct values for a respective parameter that is specific for the canvas category ‘Value Proposition’. Besides these two dimensions, other aspects like the research dependency influence the cluster analysis. As the details of the cluster technique are out of focus for this article, we refer the interested user to the respective publication for a complete discussion [17].

Taking the actual model characteristics into account, the cluster algorithm computes the distance to each cluster center using a specific calculation function. The investigated business model is assigned to the cluster with smallest distance.

As a result, the Business Model Developer provides the modeler some descriptive information on what the result of the cluster analysis means as well as a diagram overview of derived clusters. Fig. 5 shows an example of this analysis in form of a matrix representation along with a membership ranking that reflects the calculated distances to the respective cluster center. The four different clusters within the matrix have been given descriptive names and each of them is selectable. When selected by the user, the tool provides him with detailed information on the respective cluster, which has been obtained from the data on business models belonging to this cluster. This information spans a depiction of cluster-typical customer segments as well as management recommendations that describe the potential of the strategic orientation as well as possible steps towards new customer segments by means of realignment of the business model. Fig. 5 shows an example.

The user is able to create multiple variations of his business model and assess the impact of changes by repeatedly applying the cluster analysis. This way, the user can simulate alternatives of the model characteristics and identify the steps that need to be taken in order to achieve realignment of the current model.


Business Model "Boutique"

Management recommendations:

- Continuous development of research expertise
- Continuously check opportunities of joint research programs with industry and university partners
- Continuous development of service activities and direct commercialization of these
- Further strategic integration of services in business offers to improve customer satisfaction and long-term customer relationship
- Check existing opportunities for additional applications in other fields, e.g. drug development prior to and at clinical studies, to determine dose, and effectiveness of treatment.
- Due to the strong independent research focus, especially products and services for drug development could be an interesting opportunity
- The concentration on laboratory physicians and hospitals holds advantages to establish strong customer relations. However, other opportunities should be considered
- Check if non-medical customers, e.g. in food and nutrition's could be of interest

Customer segments in this cluster:

Companies from other sectors 2%
End consumers (non-medical) 0%
Other research institutions 5%
Patients 2%
Diagnostic companies 3%
Pharmaceutical companies 2%
Hospitals 29%
Laboratory Doctors 40%
Other Doctors 17%
Close

Fig. 5. Screenshot of information provided for a specific cluster

3.5 Continuous Development and Evaluation

The Business Model Developer has been developed by means of an agile development process focusing on continuous extension and improvement based on an early prototype. The overall approach has been aligned to our experience based on Extreme Model-Driven Development (XMDD) [5], which demands continuous integration of the application expert. Hence, the tool development has been aligned to regular feedback from project partners and industry experts in order to rapidly implement change requests and to avoid undesirable development in an early state.

In the course of the DPM project, feedback on the Business Model Developer has been collected from various sources, spanning regular meetings of the project's steering committee, expert interviews as well as tool presentations at healthcare-related exhibitions. In general, we observed great interest in the Business Model Developer and a low barrier to entry even for users lacking experience in business model design. The clear structure of the tool and in particular the guided modeling with the Wizard allows for the design of first business models in an immediate and intuitive manner. Having created a first model, the more sophisticated functionality of the tool can be discovered in subsequent steps as it has been integrated in an unobtrusive manner.

Besides change requests, we could identify interesting ideas and suggestions for useful extensions of the Business Model Developer, which could not be addressed anymore during the DPM project. Most of them are related to the integration of related concepts regarding strategic planning as well as to the operationalization of business models. Based on these suggestions we have identified potential for future extensions to be considered in terms of further development.

4 Conclusion and Future Work

With the Business Model Developer we have introduced a tool for the design and analysis of business models that leverages the advantages of domain-specific setups based on typed model components and structured parameters with predefined value sets. We argued that this paves the way for facilitating syntactical correctness at model design time as well as for the application of model analysis techniques.

We have applied it to the business models by the DPM industry partners, suggesting that it can meet the real-life requirements regarding business model creation and analysis. Validation and evaluation studies in other practical domains have still to come.

Two directions of ongoing important extension address the flexibilization and ad-hoc adaptation of the canvas partitioning, and the connection to the business process modeling for the process-driven development that comes with the XMDD philosophy.

4.1 Customizable Canvas Partitioning

Currently, the Business Model Developer adopts the nine canvas categories of the BMC. However, the presented approach spanning a domain-specific tool setup is not restricted to the number of categories. This leads to the idea of extending the presented approach by including the actual partitioning of the canvas into the domain-specific tool setup. Besides the library of building blocks and the category-specific parameters the categorization itself might form a third variation point in tailoring a modeling tool towards a specific application area. This especially addresses the need for an additional canvas category that has repeatedly been expressed by business experts we have spoken to. Customizable canvas categories would enable slight variations of the BMC-based approach in an immediate manner. But from a wider perspective, such a tool would not be limited to business modeling. The more abstract approach can be used for arranging a pre-determined set of domain-specific components and create relations between them on a customizable canvas in terms of its partitioning into distinct categories. This would open up new opportunities in various application fields.

4.2 Application of Process-Driven Development

The cluster analysis described in the previous section is an excellent example of the tool's core logic. In general, analysis techniques might comprise sophisticated calculations based on complex decisions relying on various aspects of the actual model state. Such techniques may be developed by business analysis experts and implemented into software tools by development staff. This is the point at which the domain expert loses control over the development process; from the user's perspective it is not traceable whether the implementation of the algorithm is correct or not.

Applying model-driven development utilizing the jABC framework represents an alternate approach that keeps the domain expert connected and involved into the development process at any time. Encapsulated services, which are the basic model entities by means of the jABC framework, can be generated from the domain-specific setup of the Business Model Developer using the Genesys compiler [18]. These services can be used within the jABC to create executable process models representing the desired analysis process. Finally, code can be generated from these process models and integrated into the Business Model Developer to be invoked for the analysis of created business models. This model-driven approach simplifies the design of model analysis processes that are tailored towards the respective domain-specific modeling environment of the Business Model Developer.

References

1. Osterwalder, A., Pigneur, Y.: Business model generation. John Wiley & Sons, Inc. (2010)
2. Kühne, T.: Matters of (Meta-) Modelling. Software and Systems Modeling 5, 369–385 (2006)
3. Margaria, T., Steffen, B.: Simplicity as a Driver for Agile Innovation. Computer 43(6), 90–92 (2010)
4. Floyd, B.D., Boßelmann, S.: ITSy - Simplicity Research in Information and Communication Technology. IEEE Computer 46(11), 26–32 (2013)

5. Margaria, T., Hinchey, M.: Simplicity in IT: The Power of Less. IEEE Computer 46(11), 23–25 (2013)
6. Margaria, T., Steffen, B.: Service-Orientation: Conquering Complexity with XMDD. In: Hinchey, M., Coyle, L. (eds.) Conquering Complexity. Springer (2012)
7. Margaria, T., Steffen, B.: Continuous Model-Driven Engineering. Computer 42, 106–109 (2009)
8. Steffen, B., Margaria, T., Nagel, R., Jörges, S., Kubczak, C.: Model-Driven Development with the jABC. In: Bin, E., Ziv, A., Ur, S. (eds.) HVC 2006. LNCS, vol. 4383, pp. 92–108. Springer, Heidelberg (2007)
9. Osterwalder, A.: The Business Model Ontology - A Proposition in a Design Science Approach. PhD Thesis, Universite de Lausanne (2004)
10. Kaplan, R., Norton, D.: The balanced scorecard: Measures that drive performance. Harvard Business Review 70(1) (1992)
11. Porter, M.E.: Competitive advantage. Free Press, New York (1985)
12. Freeman, R.E.: Strategic management: A stakeholder approach. Pitman, Boston (1984)
13. Strategyzer, `http://www.strategyzer.com`
14. Guarino, N.: Formal ontology, conceptual analysis and knowledge representation. International Journal of Human and Computer Studies 43(5/6), 625–640
15. Motik, B., Patel-Schneider, P.F., Parsia, B., Bock, C., Fokoue, A., Haase, P., ... Smith, M.: OWL 2 Web Ontology Language: Structural Specification and functional-style Syntax. W3C Recommendation 27, 17 (2009)
16. Project "Service Opportunities in Personalized Medicine", `http://dpm.ceip.de`
17. Kamprath, M., Halecker, B.: A Systematic Approach for Business Model Taxonomy - How to Operationalize and Compare large Quantities of Business Models? In: 5th ISPIM Innovation Symposium (2012)
18. Jörges, S. (ed.): Construction and Evolution of Code Generators. LNCS, vol. 7747, pp. 3–221. Springer, Heidelberg (2013)

Dr. Watson? Balancing Automation and Human Expertise in Healthcare Delivery

Mark Gaynor[1], George Wyner[2], and Amar Gupta[3]

[1] Saint Louis University, School for Public Health and Social Justice, St. Louis, MO, USA
mgaynor@slu.edu
[2] Boston College, Carroll School of Management, Chestnut Hill, MA, USA
george.wyner@bc.edu
[3] Pace University, Seidenberg School of Computer Science and Information Systems, New York City, NY, USA
agupta@pace.edu

Abstract. IBM's Watson, the supercomputer that beat two Jeopardy champions in a televised competition, has inevitably engendered speculation about what this surprising performance bodes for the role of computers in the workplace. How can Dr. Watson be best utilized in medicine and clinical support systems? This paper defines Computer Enhanced Medicine technology for healthcare as any medical application where the required tasks are split between a human being and a computer-controlled device.

Keywords: Clinical Support System, Computer Enhanced Medicine, Telehealth.

1 Introduction

IBM's Watson, the supercomputer that beat two Jeopardy champions in a televised competition, has inevitably engendered speculation about what this surprising performance bodes for the role of computers in the workplace. The inevitable question, asked of any newly arrived celebrity, is "Watson, what are you going to do next?" As reported in the New York Times [1], the answer provided by IBM is: Watson will be taking on healthcare. The New York Times adds that IBM's Watson, which uses statistical natural-language processing techniques [2] is working with oncologists at Cedars Sinai Cancer Institute and is also being trained as a medical student at the Cleveland Clinic. This seems like a timely choice, given that despite many advances in medicine, healthcare remains a labor-intensive field that is fraught with costly mistakes, which is one reason that healthcare costs have soared over time. It has been harder to increase productivity in this sector because the delivery of healthcare includes hard-to-replace human components. However, advances in computing and communications technology, of which Watson is only the latest, are altering the global healthcare landscape by providing application and infrastructure designers with new technology that has the potential to reduce costs and improve outcomes. These new technologies can:

- Enable detailed analysis of healthcare data to elicit underlying trends and interrelationships; for example, analysis of mammography data from a heterogeneous patient group over a geographically diverse region;
- Facilitate transmission and integration of healthcare records, such as the sharing of pre-hospital electronic medical records;
- Enable healthcare professionals to render assistance to patients separated by significant geographic distances from each other, such as remote computer enhanced diagnosis of radiological images.

We refer to these applications of technology to healthcare as Computer Enhanced Medicine (CEM) which we define as any medical application in which the tasks required to accomplish the application are split between a human being and a computer controlled device, and for which this combination can potentially yield better outcomes than can be obtained by humans alone or computers alone. This definition includes many types of telemedicine, client-server, beacon alert type services, and peer-to-peer applications. These applications can range from complex, such as providing real-time support for emergency trauma patients based on transmission of real-time vital signs, to relatively simple, such as translating and scanning paper/film based medical records into electronic form.

While CEM has great promise, the challenge remains in how to incorporate computers into healthcare delivery processes where human judgment has traditionally been deemed to be essential to so many tasks.

Watson's victory illustrates that even applications based on the latest advances in technology can require human intervention. While Watson's performance in Jeopardy was impressive, it was glaringly off-base in its choice of "Toronto" as the response to this clue about a US City: "Its largest airport is named for a World War II hero; its second largest for a World War II battle." In Watson's favor, one must acknowledge that there is a Toronto, Kansas (and that Watson's second choice was actually the correct answer, Chicago), and also that Watson indicated its lack of confidence in its own answer by appending several question marks. However, this does reinforce the sense that in any domain where the cost of a single very wrong answer is high, human judgment must continue to be in the loop, and clearly healthcare is such a domain.

One example of this issue can be found in radiology. When a human radiologist looks at a mammogram, there is a significant chance for a false positive or a false negative in terms of interpreting the mammogram and diagnosing potential problems. New computer-based techniques for "reading" digital mammograms can be used to reduce the potential probability for false positives and false negatives. However, a human radiologist must review the results generated by the computer in order to avoid the risk of costly computer errors due to obvious and subtle errors that a trained human radiologist can quickly spot. Such need for human involvement in several steps in the healthcare delivery process poses a challenge to the deployment of new technologies.

We believe that a part of the difficulty is the focus placed by system design methodologies on automating existing processes. Liu and Wyner [3] refer to this as the “automation approach,” which consists of examining the individual activities in a process to identify those amenable to automation or some form of computer support. As such, this approach is dependent on how the process to be supported is conceptualized. For example, an automation approach to designing a system to prescribe drugs might focus on quick data entry and a paperless workflow, but if one takes into account the possibility of connectivity to an Electronic Medical Record (EMR) and a current database of known drug interactions, then one can reframe the process to include enhanced and automated checking for interactions between the drug being prescribed and those already taken by the patient. Often the most dramatic benefits of computer-enhanced work depend on significant, in some cases radical, changes in the underlying business process.

In the case of healthcare, the automation of existing processes will be constrained by the presence of tasks that require a skilled professional (e.g., physician, nurse, or pharmacist) and such tasks must be taken off the table in any discussion of computer-based process enhancement.

Our premise is that this characteristic of healthcare delivery processes, while being a limiting factor, also serves as a basis for exploring possible process redesign to enable more effective use of IT. The problem, as we see it, is that the automation decision has been largely attempted at too coarse grained a level: replacing an existing human activity in the process by an entirely automated activity. We believe a more useful approach would be to take a finer grained view of the existing process. Specifically, analysts should: (i) closely examine each task that appears to require manual action by a healthcare professional and (ii) identify the sub-tasks that comprise this task.

By breaking down a single manual task, one may be able to identify sub-tasks that can be fully or partially automated without removing critical human involvement in the task as a whole. As we shall illustrate, this approach can provide a way to integrate new computer technologies into an existing healthcare process by redesigning critical activities in the process to allow for a sensible division of labor among skilled professionals and advanced information technologies. By subdividing tasks in a manner that facilitates adoption of emerging computer and communications technology, we can reap the potential benefits of CEM, including the intelligent use of technology to improve patient care while reducing costs, and the efficient allocation of scarce human resources by taking advantage of technology.

2 A Methodology for Balancing Automation and Human Expertise

We have developed a new approach, PAVDOT (Partial Automation Via Decomposition of Tasks), to provide analysts with guidance on how to uncover these additional opportunities for integrating new information technologies into the healthcare delivery process. PAVDOT was developed based on insights gained by two of the authors over several research projects, some of which are discussed in the examples below. PAVDOT is a distillation of the insights gained from actual experiences.

PAVDOT consists of the following steps:

1. **Define Scope**: Choose area(s) for enhancement. One can use an established framework such as Balanced Scorecard, Six-Sigma, Lean Value Stream, or Value Chain Analysis. However, a more informal approach can be used, provided that it results in a principled decision on where the organization should focus its system enhancement efforts. For example, one might simply ask stakeholders what their pain points are. The output of this step should be a specified scope that defines the focus of the steps that follow.
2. **Identify Key Activities**: Identify the key activities associated with the processes to be enhanced. While this can be done using a process-mapping technique such as a flow chart, activity diagram, or BPMN, an elaborate process model is not always necessary or even helpful. What is needed is a list of activities that can be considered for decomposition (in step 4 below). An additional output of this step may be the surfacing of some requirements for the process system redesign. Such requirements can be captured for later use in the subsequent design process.
3. **Assemble Technology Toolbox**: Survey enabling technologies for opportunities to enhance or automate parts of the process. Examples include Neural and Bayesian Networks from the machine learning area of AI, cloud and high-speed broadband networking in the distributed computing and networking area, and inexpensive massive storage along with an excess of CPU cycles in the systems area. Note that the above examples are deliberately heterogeneous in nature, including long established mainstream technologies and newer techniques, as well as a mix of general approaches and specific technologies. The specific techniques to be included are going to depend on the specific project and the results of the survey of enabling technologies. The resulting set of possible technologies will serve as a technology "tool box" from which analysts can select appropriate technologies in the process redesign that is conducted in step 4.
4. **Redesign Process by Partial Automation**: Decompose one or more activities (identified in step 2 above) to allow an existing manual task to be distributed among both human actors and the technologies identified in step 3. This key re-design step depends not only on the technology toolbox, but also on institutional policies governing authorization, and the nature of the overall task. Assign tasks to human or automated actors as appropriate. In this step, tasks may need to be adapted to the capabilities of technologies, and new applications based on those technologies may need to be developed or adapted to the requirements of the specific tasks to be automated. One example of this notion is the migration to interoperable Electronic Medical Records (EMR). EMRs change the process flow in most organizations and create new opportunities for applications such as checking for adverse drug interactions and allergies when electronically prescribing medication. The output of this step is a set of changes to the existing process, consisting of newly defined activities and a plan to automate some of those activities using specific technologies, which in turn may result in changes to the remaining manual activities.

Together the four steps of PAVDOT are intended to help generate new alternative task decompositions that might not otherwise be considered. The core of our proposal is contained in step 4, which directs the analyst to develop a fine-grained view of the process under discussion.

3 Radiology Example

The potential value of PAVDOT is analyzed by illustrating how it might be applied in radiology. Teleradiology is one form of telemedicine with a long history [4] and a solid business case with enough patients [5]. The important issues for debate have shifted from technical concerns about transmission speed and image compression to more management related issues such as governance, medico-legal issues and quality assessment [6]. For radiology we expect PAVDOT to help generate ideas about different ways change the process of patient care.

3.1 Technology-Based Remote Diagnosis of Radiology Images (TDRI)

Technology based remote diagnosis of radiology images (TDRI) is a type of teleradiology that involves remote interpretation of the medical image by a computer based application.

Current trends in healthcare are dictating the growth of teleradiology. First, there is a growing shortage of radiologists, because of a significant number of radiologists retiring from practice and training programs not keeping pace with growing demand. Second, the aging population and the advent of newer imaging technologies are leading to annual increases in imaging volumes; for example, a 13% increase in the utilization of radiological imaging was observed among Medicare beneficiaries. Third, the increased use of imaging technologies in trauma situations has led to a corresponding need for round-the-clock radiological services in hospital emergency rooms. The concept of Technology-based Diagnosis of Radiology Images offers the potential to use the scarce resource of radiologists in a more efficient manner. Kalyanpur et al proposed a service delivery model for TDRI that coordinates both workflow and payments [7].

Mammography is an example of a teleradiology sub-specialty that can benefit from intelligent computer technology. One out of eight women in the US will develop breast cancer during her lifetime. Early detection is a woman's best weapon against breast cancer, which is 97% curable when detected and treated in the early stages. The mammogram is the gold standard for screening breast cancer. With the trend towards people living longer lives and taking proactive measures for their health, the demand for mammograms is increasing at a significant pace. The number of mammograms performed each year is rapidly increasing because of three factors: (i) More women in the traditional age group are undergoing mammograms; (ii) The mammograms are performed more frequently than before; and (iii) The recommended age for conducting mammograms has been gradually lowered by medical agencies. All these factors increase the aggregate work on the radiologists.

The issue of errors in mammography (both false positives and false negatives) has been studied in detail by many researchers. Estimates are that radiologist miss 10% - 20% of the cancers currently detectable by a screening mammogram, which allows the disease at least another year to progress. There is a high degree of liability for radiologists due to missed diagnoses. To mitigate this problem, some radiology screening centers employ two radiologists to read each case. This approach involves significant cost to support an additional radiologist, reduces the number of total mammograms that can be performed within a center, and is problematic due to the shrinking numbers of radiologists in the field of mammography, especially in the U.S.

We describe how PAVDOT could be applied to this domain with the following steps:

- **Step 1:** Define Scope: Given the preceding discussion, the important area to address in our analysis would be how to redesign the process of reading mammograms to optimize the productivity of radiologists, reduce the incidence of errors (both false positives and false negatives), and reduce the costs incurred in performing mammograms. This will be the scope for our analysis.
- **Step 2**: Identify Key Activities: The key activities in the traditional mammography process might be listed as follows:

  1. Order. The primary care physician orders the test.
  2. Image. An analog or digital Mammography image is created.
  3. Diagnose. The image is evaluated, taking into account the patient's medical history.
  4. Report. The radiologist creates a report of the diagnosis.

- **Step 3**: Assemble Technology Toolbox: Technologies that exist or are emerging with the potential to enable CEM innovation in mammography include:

  1. Inexpensive access to large amounts of storage for digital images.
  2. Availability of fast networks to transfer large images.
  3. Access to longitudinal data of mammogram images across a broad range of the population.
  4. Availability of Artificial Intelligence algorithms such as neural networks that can learn how to predict relationships in unstructured data.
  5. Specific technologies based on the above. For example, the Portuguese system developed by INESC Porto and FEUP has been 100% effective in detecting malignant tumors (www2.inescporto.pt/ip-en/news-events/press-releases/ software-portugues-possibilita-analise-automatizada-de-mamografias).

  Our goal in identifying these technologies is to choose a range of important enabling technologies both general and quite specific (item 5 on the list) in order to stimulate thinking about the widest range of options.

- **Step 4:** Redesign Process by Partial Automation: Based on the technologies above, the task most likely to benefit from partial automation would be activity 3, the Diagnose activity. Considering that two radiologists are sometimes employed, we consider decomposition of this activity into two sub-activities: Suggest Diagnosis (The image is evaluated here) and Confirm Diagnosis (The image is evaluated, taking into account both the diagnosis suggested and the patient's medical history).

We observe that a partial automation strategy might be to assign the first of these two activities to a computerized system, allowing a radiologist to confirm the suggested diagnosis.

This would yield the following amended process:

1. Order. The primary care physician orders the test.
2. Image. A digital Mammography image is created.

3a. Suggest Diagnosis. The image is evaluated automatically using Computer Aided Detection (CAD) techniques.

3b. Confirm Diagnosis. A radiologist, taking into account both the diagnosis suggested in activity 3a and the patient's medical history, evaluates the image.

4. Report. The radiologist creates a report of the diagnosis.

The primary task in this process is diagnosis: a radiologist or computer-based application examines the image in order to diagnose the patient. The decomposition of this single task into two sub-tasks, as discussed above, allows a technology-based solution to suggest a diagnosis that can be confirmed by a certified radiologist. The sub-division of the diagnosis task allows for a more flexible use of technology by permitting some, but not all of the diagnosis effort needs to be automated. Note that the computerized diagnosis is based on the image alone, whereas the radiologist takes the patient's full medical record into account. This reflects the current limitations of CAD technology. However, by having the image-only reading precede the radiologist's diagnosis, significant benefits can be achieved.

4 Discussion

This use of Computer Aided Detection (CAD) techniques in mammography could mitigate the problem of the growing shortage of radiologists, as well as reduce or eliminate many of the instances of missed symptoms. By using a CAD-based approach in conjunction with a human radiologist, one essentially attains the scenario of two independent readings of each mammogram, with the human radiologist being actively involved in the process and making the final determination in all cases. The advantages of this approach are: (i) The capital investment of using a CAD service is significantly less, when compared to that of employing a second radiologist; (ii) Current and previous cases can be made available to the radiologist on-line for necessary comparison; (iii) Results and information can be made available anywhere via the Internet; and (iv) Improvements to the algorithm and core technology can be readily disseminated. Further, the suggested approach reduces the incidence of second visits and the level of patient anxiety, by providing expert (specialty) second opinions when needed in a timely manner through a teleradiology model. This approach was originally proposed in 2001 and 2002 [8].

There are several problems with the automatic evaluation of radiologic images. Once a computer algorithm suggests a particular diagnosis it might create a bias towards agreement. This can lead to increased false positives, which can cause extreme distress when serious conditions such as breast cancer are incorrectly diagnosed. With good user design and inclusion of features critical to success these problems can be minimized [9]. Dr. Watson does not present a diagnosis of a specific condition, but a list of possible diagnoses and their probabilities. This encourages the care provider to explore a greater breath of conditions, hopefully without biasing the decision because of the probability estimates.

General radiology and mammography differ in one important respect: mammography is more amenable to computer-aided diagnosis because of the large database of available images. As discussed above, combining a common test with a comprehensive database is well suited to computer analysis. However, as suggested by the discussion of Watson, it is unlikely that current technology will be deemed sufficient for confirming a medical diagnosis without human intervention. In this case, the sub-division of the diagnosis task into the two activities –Suggest Diagnosis and Confirm Diagnosis -- promotes a more efficient and flexible use of computer technology to improve both the speed and the quality of the diagnosis.

5 Concluding Remarks

The traditional model of healthcare required that medical personnel be in immediate proximity to their patients. Computers can not only search through millions of images of mammograms in very short periods of time in order to locate those images that match the key characteristics of the one currently under review in the clinic, but computers can learn from these vast data sets; such power is clearly beyond the capability of a single doctor or even groups of doctors.

This paper does not address the many regulatory, legal, and privacy issues concerned with exchanging personal health care data. For example, in the United States where States regulate medical certification it has been difficult for tele-health to cross state lines. One bright exception is Veterans Affairs that allows it's care providers to practice across State lines.

Learning systems such as Watson hold tremendous potential to improve health outcomes while reducing costs. To reach the full potential of CEM the technology must be intelligently applied. Watson's mixed performance on Jeopardy is a reminder that humans must review any decision that has a high cost of error.

This technology evolution has the potential to create exciting possibilities with CEM because it enables faster, higher quality, and more cost effective medical services by applying intelligent technology to tasks that are traditionally accomplished without computer applications. The Mammography diagnosis case study is a good illustration of this. Using emerging computing technology to suggest abnormalities and a list of possible causes based on a comprehensive database will find trends that are hard for humans to discover.

In evaluating the potential of PAVDOT, we must point out that the example above has been simplified and is an illustration of how PAVDOT might be applied, not a report of an actual case study of PAVDOT, which is still in its early stages of development. That said, however, the example suggests the potential value of an approach like PAVDOT and suggests how it might be applied to generate new alternatives for combining human and computational resources in order to improve outcomes and reduce costs. It is important to note that PAVDOT is focused on helping analysts and stakeholders to generate new "partial automation" alternatives for consideration. It does not evaluate such alternatives nor does it take into consideration issues like change management, nor is it intended to serve as a design methodology. That said, we believe that PAVDOT does have the potential to nudge analysts and other stakeholders in the direction of better aligning work flow with computer technology, and that this is an important potential benefit worthy of further investigation.

Our preliminary experience with PAVDOT suggests a possible future trend in the design of CEM. We predict that healthcare will increasingly use a portfolio approach comprised of three closely-coordinated components seamlessly interwoven together: healthcare tasks performed only by humans, healthcare tasks performed by combining humans with computers, and healthcare tasks performed by computers without direct human involvement. We are hopeful that this three-pronged approach can lead to better and more cost-effective healthcare.

References

1. Lohr, S.: Software Assistants for doctors Are Making Progress. New York Times (2013)
2. Edwards, C.: Using Patient Data for Personalized Cancer Treatments. Communications of the ACM 57(6), 13–16 (2014)
3. Liu, X., Wyne, G.: Coordination Analysis: A Method for Deriving Use Cases from Process Dependencies. In: 4th International Conference on Design Science Research in Information Systems and Technology, DESRIST 2009 (2009)
4. Goldberg, M.: Teleradiology and Telemedicine. Radiologic Clinics of North America 34(3), 647–665 (1996)
5. Bergmo, T.: An Economic Analysis of Teleradiology Versus a Visiting Radiologist Service. Journal of Telemedicine and Telecare 10, 310–314 (2003)
6. Binkhuysen, B.: Teleradiology: Evolution and Concepts. European Journal of Radiology 78(2) (2010)
7. Kalyanpur, A., Neklesa, V., Pham, D., Forman, H., Stein, S., Brink, J.: Implementation of an International Teleradiology Staffing Model. Radiology 232 (2004)
8. Gupta, A., Norman, V., Mehta, V., Benghiat, G.: Contata Health Inc. (2002)
9. Kawamoto, K.: Improving clinical practice using clinical decision support systems: A systematic review of trials to identify features critical to success. BMJ 330, 765 (2005)

Semantic Heterogeneity in the Formal Development of Complex Systems: An Introduction[⋆]

J. Paul Gibson[1] and Idir Ait-Sadoune[2]

[1] Département Logiciels-Réseaux, Telecom-SudParis, Évry, France (SAMOVAR UMR 5157)
paul.gibson@telecom-sudparis.eu
[2] Supelec, Gif Sur Yvette, France
idir.aitsadoune@supelec.fr

System engineering is a complex discipline[1], which is becoming more and more complicated by the heterogeneity of the subsystem components[2] and of the models involved in their design. This complexity can be managed only through the use of formal methods[3]. However, in general the engineering of software in such systems leads to a need for a mix of modelling languages and semantics; and this often leads to unexpected and undesirable interactions between components at all levels of abstraction[4]. There are currently no generally applicable tools for dealing with this heterogeneity of interactions in the engineering of complex systems.

The heterogeneity exists in 3 different dimensions:

– Abstraction — as software engineers move from requirements to implementation, the semantics of the modelling languages move from the problem domain to the solution domain. Thus, it is quite common to see two or more languages used as the modelling moves from the abstract to the concrete (from the non-operational to the operational)[5].
– Systems of systems — software should not be isolated from the system and environment in which it is intended to operate. Systems are now engineered from components including software, hardware, wet-ware, etc .... The types of these subsystems can vary greatly: synchronous or asynchronous, deterministic or non-deterministic, parallel or sequential, etc .... It is unlikely that a single language is best suited to modelling such heterogeneity[6].
– Synthesis and analysis — the language in which one models a system is not usually the same language in which one reasons about the relationship between models, and the correctness of one model with respect to another[7].

There needs to be a clear separation of concerns, in these 3 dimensions, in order to facilitate re-usable models, methods, tools and software processes (methodologies). There also needs to be a simple way of integrating the different concerns.

⋆ This work was supported by grant ANR-13-INSE-0001 (The IMPEX Project http://impex.gforge.inria.fr) from the Agence Nationale de la Recherche (ANR).

As with object oriented architectures, low coupling and high cohesion are strong indicators of good design[8]. In a formal approach to system engineering we need low coupling between our different modelling languages and high cohesion within them. This can be best achieved by formal specification of good interfaces between the different types of semantics. Currently, the state of the art in heterogeneous system modelling is away from an ideal development environment, where the interfacing (between different semantic models) would be automated. This is the long-term objective, but we have only recently embarked on the journey towards this goal.

A previous thematic track initiated the research in the direction of the problems arising due to the heterogeneous nature of formal modelling[9] . Two of the published papers illustrated different techniques for managing the heterogeneity. In *Leveraging Formal Verification Tools for DSML Users: A Process Modeling Case Study*[10], we see a model driven development approach where formal methods are used to translate between different modelling languages. In *An Ontological Pivot Model to Interoperate Heterogeneous User Requirements*[11] we see a pivotal ontological model being used to manage heterogeneity of vocabularies and heterogeneity of formalisms during requirements modelling.

In this thematic track, we emphasis the complex nature of systems engineering and the need for automated tool support for integrating different semantic models. We note that the accepted papers discuss not only theoretical aspects, but also hint at methodological aspects which are key to industrial transfer of these approaches. In *Modelling and Verifying an Evolving Distributed Control System Using an Event-based Approach*[12] we see component-based system engineering where abstraction plays a key role in permitting the integration of different component types (specified using different semantic models), and reasoning about their dynamic interaction. In *Requirements driven Data Warehouse Design: We can go further*[13] we see that ontological reasoning mechanisms can used to automatically construct a set of requirements that are coherent and non-conflictual, even when expressed in a variety of modelling languages. Finally, the paper *On Implicit and Explicit Semantics: Integration issues in proof-based development of systems*[14] illustrates how formal ontologies can be used to model re-usable domain knowledge in an explicit manner, and how this knowledge can be used to prove the correctness of a system that operates within the domain environment. Refinement of the system model can then be used to guarantee correct construction of a system within the specified context.

The track demonstrates that progress is being made in this very challenging area. However, much more is left to do.

References

1. Stevens, R., Brook, P.: Systems engineering: coping with complexity. Pearson Education (1998)
2. Eker, J., Janneck, J.W., Lee, E.A., Liu, J., Liu, X., Ludvig, J., Neuendorffer, S., Sachs, S., Xiong, Y.: Taming heterogeneity-the ptolemy approach. Proceedings of the IEEE 91(1), 127–144 (2003)

3. Balarin, F., Lavagno, L., Passerone, C., Sangiovanni-Vincentelli, A., Sgroi, M., Watanabe, Y.: Modeling and designing heterogeneous systems. In: Cortadella, J., Yakovlev, A., Rozenberg, G. (eds.) Concurrency and Hardware Design. LNCS, vol. 2549, pp. 228–273. Springer, Heidelberg (2002)
4. Gibson, J., Mermet, B., Méry, D.: Feature interactions: A mixed semantic model approach. In: McGloughlin, H., O'Regan, G. (eds.) 1st Irish Workshop on Formal Methods (IWFM 1997). Electronic Workshops in Computing, Dublin, Ireland, BCS (July 1997)
5. Hazzan, O., Kramer, J.: The role of abstraction in software engineering. In: Companion of the 30th International Conference on Software Engineering, ICSE Companion 2008, pp. 1045–1046. ACM, New York (2008)
6. Baldwin, W.C., Sauser, B.: Modeling the characteristics of system of systems. In: IEEE International Conference on System of Systems Engineering, SoSE 2009, pp. 1–6. IEEE (2009)
7. Adrion, W.R., Branstad, M.A., Cherniavsky, J.C.: Validation, verification, and testing of computer software. ACM Computing Surveys (CSUR) 14(2), 159–192 (1982)
8. Briand, L.C., Wüst, J., Daly, J.W., Victor Porter, D.: Exploring the relationships between design measures and software quality in object-oriented systems. Journal of Systems and Software 51(3), 245–273 (2000)
9. Ait-Ameur, Y., Méry, D.: Handling heterogeneity in formal developments of hardware and software systems. In: Margaria, T., Steffen, B. (eds.) ISoLA 2012, Part II. LNCS, vol. 7610, pp. 327–328. Springer, Heidelberg (2012)
10. Zalila, F., Crégut, X., Pantel, M.: Leveraging formal verification tools for dsml users: A process modeling case study. In: Margaria, T., Steffen, B. (eds.) ISoLA 2012, Part II. LNCS, vol. 7610, pp. 329–343. Springer, Heidelberg (2012)
11. Boukhari, I., Bellatreche, L., Jean, S.: An ontological pivot model to interoperate heterogeneous user requirements. In: Margaria, T., Steffen, B. (eds.) ISoLA 2012, Part II. LNCS, vol. 7610, pp. 344–358. Springer, Heidelberg (2012)
12. Attiogbé, C.: Modelling and verifying an evolving distributed control system using an event-based approach. In: Margaria, T., Steffen, B. (eds.) ISoLA 2014, Part II. LNCS, vol. 8803, pp. 573–587. Springer, Heidelberg (2014)
13. Khouri, S., Bellatreche, L., Jean, S., Aït-Ameur, Y.: Requirements driven data warehouse design: We can go further. In: Margaria, T., Steffen, B. (eds.) ISoLA 2014, Part II. LNCS, vol. 8803, pp. 588–603. Springer, Heidelberg (2014)
14. Ait-Ameur, Y., Gibson, J.P., Méry, D.: On implicit and explicit semantics: Integration issues in proof-based development of systems. In: Margaria, T., Steffen, B. (eds.) ISoLA 2014, Part II. LNCS, vol. 8803, pp. 604–618. Springer, Heidelberg (2014)

Modelling and Verifying an Evolving Distributed Control System Using an Event-Based Approach

Christian Attiogbé

LINA - UMR CNRS 6241 - University of Nantes
F-44322 Nantes Cedex, France
`Christian.Attiogbe@univ-nantes.fr`

Abstract. Evolving distributed systems are made of several physical devices distributed through a network and a set of functionalities or applications hosted by the physical devices. The configuration of the physical components may be modified through the time, hence the continuous evolving of the whole system. This should affect neither the hosted software components nor the global functionning of the whole system. The components of the systems are software components or physical components but their abstract models are considered with the aim of modelling and reasoning. We show that an event-based approach can be benefically used to model and verify this kind of evolving control systems. The proposed approach is first presented, then the CCTV case study is introduced and modelled. The resulting model is structured as a B abstract machine. The functional properties of the case study are captured, modelled and proved. The refinement technique of Event-B is used to introduce and prove some properties.

Keywords: Heterogenous components, Modelling, Event-B, Property verification.

1 Introduction

Many software applications are required for decentralized control of highly interacting components; they need to be not only reliable but also extensible hence the use of *evolving distributed control systems.*

Context. The study of distributed systems has been the subject of years of research and development [6,5,9]; several results exist at operating systems, middleware and application levels. However specific attention was paid to static distributed environment where the involved hosts (client and server processes or peers), are known and clearly identified as well as their architecture or configuration. Three main architectures have governed the design of distributed applications: two-tier with the interaction between a client and a server, three-tier with two levels of interaction (client-server, server-host) and n-tier architectures. The later is the more general one: a client process located on any host may interact via a middleware with one or several distributed servers.

The client/server two-tier architecture, generalized to n-tier and improved with the peer-to-peer architecture, is used to structure most of the distributed applications.

However, when the environment is not static but dynamic, ie its architecture is evolving according to the (re)configuration of the hosts, then additional difficulties should be considered, more precisely at the system design level. These difficulties rely on the identification of the hosts, the structuring of the exchanged messages, the dynamic aspect of the links and consequently the structure of the overall architecture of the system. Notably, instead of addressing a request to a given server, a thin client will address a request to its environment. There is a shift from the client-server relationship to a more flexible relationship between an application and its virtual environment. Modelling and analysis should not consider the precisely known interacting entities but their virtual counterpart.

Motivation. The current keen interest for virtualized distributed environment (aka *cloud, grid*) pushes a difficulty at application modelling level because, more and more applications are dedicated to *evolving* distributed environment, to store data, to request functionalities or services, etc. Evolving distributed systems are those with an adhoc highly changing architecture, due to the behaviour of their components. Heterogeneity of components (physical devices, software, various models) is a specific feature of these systems. In this work we target such evolving distributed systems and more specifically modelling and reasoning on a system which will be implemented or deployed as an evolving dynamic distributed system. That means the interacting processes are neither explicitly known nor explicitly identified, but the expected behaviour should be described and formally analysed. This is a key to mastering heterogeneity.

Contribution. We propose an event-based approach to make it easy the modelling and the analysis of evolving distributed applications. We show the effectiveness of the proposed approach on a case study: a CCTV[1] control system which may evolve depending on the used devices and their reconfiguration. The approach is based on the use of a virtual network of processes, an event-based modelling and refinement; it can be exploited as a pattern in many other similar cases.

Organisation. The remainder of the article is structured as follows. Section 2 is devoted to the evolving systems considered in the paper; their specificities are underlined and then we present the CCTV case study (Sect. 2.2). Section 3 and Section 4 are the core of the paper. We present the used approach (Sect. 3.2) and its application to the modelling and verification of the CCTV case study (Sect. 4). Finally, Section 5 concludes the article with some lessons and future work.

[1] Closed-Circuit Television.

2 Evolving Distributed Systems

2.1 Issues on Evolving Distributed Systems

The correction of distributed systems is still a challenging concern according to modelling, verification and implementation; this is essentially due to mastering complex non-deterministic behaviours, concurrency and safety. Moreover the adoption of the *service-oriented* paradigm enhances not only the need of assistance methods and techniques to build distributed systems but also the needs of highly evolving distributed systems. Processes that represent parts of the system are dynamically linked and unlinked, their behaviours may vary in the time; consequently the global architecture of the system is changing dynamically.

One solution for resolving the difficulties is to build a formal model which serves as a basis for the rigorous analysis and for the construction of (parts of) the system.

Consider a system made of an undefined number of peers which cooperate and provide services; the peers may be mobile, linked and unlinked to different other peers. New peers may be involved with respect to needed functionalities. The architecture of the system is therefore continuously changing.

The message-passing technique with explicit naming and peers identification is not tractable in such an adhoc context; an implicit message passing is needed instead. The evolving should not prevent from maintaining the functionalities and the required properties of a distributed system. Among the issues to be addressed for this purpose, we consider:

– the modelling issue, in order to best capture the requirements and provide a faithful model. Appropriate structuring mechanisms are needed to get an extensible and open system instead of a closed one.
– the reasoning issue, in order to provide method and guidelines to study desired properties. This should be achievable in parts of the global formal model.

We use a real life CCTV case study to present our contribution. This kind of system is often distributed and highly evolving due to the evolution of the architecture of the devices to be controlled.

2.2 The CCTV Control System

A CCTV[2] system is often used for the surveillance of industrial plants, the surveillance of homes or the control of various distributed equipments from a central or a decentralized control room. There are several devices which are linked to their controllers, but this architecture is changing with respect to specific surveillance policies or the adding of new devices, the removing of existing devices. In a basic CCTV system the video captured by cameras are displayed on dedicated screens which may have their own controller.

[2] Closed-Circuit Television.

The CCTV system may evolve in such a way that, instead of displaying the video from cameras directly on screens, Digital Video Recorders (DVR) are added to record the video which are used or analyzed later for various purposes (Video Contents Analysis). Consequently, the architecture of system changes: the videos are now recorded before being displayed on the screens.

The system is made of a set of cameras linked with control screens which are under the supervision of humans. The cameras are directed towards parts of a predefined area to be controlled for surveillance or intrusion detection. The cameras send video streams to the control unit which displays the streams on activated screens. More cameras and screens may be added to cover some unreached area without changing the functioning of the system. Thus the CCTV control system is made on the one hand with several controllers, and on the other hand with control screens, control cameras and video display units. The video captured by the cameras are displayed on screens. A controller is linked with one or several (input) cameras, it displays its output on one or several screens, and it may have a DVR for storage.

The requirements stipulate that initially there is no recording of video; then it can be decided to record and save the video while displaying them. Hence a digital video recorder component will be introduced in the system. Accordingly, the video stream will not only be the input of the screens but also the input of the recorder if any.

Some functional properties are required to ensure the good functioning of the control system. We identify and name each of them in order to refer to them in the analysis section. They are summarized in Tab. 1.

Table 1. Synthesis of the functional requirements

FReq_AreaOK	*All the area to be supervised is under control*
FReq_ctrlNCam	*Each controller can manage several cameras*
FReq_cam1Ctrl	*Each camera is managed by only one controller*
FReq_DispOk	*All the captured videos are displayed on some screens*
FReq_RecDispOK	*All active cameras should be under control*
FReq_RecOK	*All video received from the cameras are recorded*
FReq_NewDev	*It should be possible to add new cameras and screens*

We have to model and analyse the functioning of this control system which *i)* is made of several controllers, *ii)* manages several different devices, and *iii)* has a varying architecture.

3 Modelling and Verification Approach

To master the dynamic aspect of evolving systems and the heterogeneity of their components, abstract models with a light composition approach are required. One solution is to view the systems as a virtual net of components; the components may share abstract channels for communication. At a more concrete level of each component, independently from the other components, the abstract channels may be implemented in various way.

3.1 The Core Modelling Approach

Our approach is based on an event-based composition as a weak coupling of processes that will interact through a common state space that includes abstract channels dedicated to message passing. Our initial work was introduced in [3]. The model of a global system consists of

- a global state space made of the data types identified in the requirements;
- a set of *process types* that describe the identified components of the global system. The component may be physical or logical. We have to identify which are the interacting processes and the messages that they exchange. Each process has its state space, a part of which is shared with the global state space;
- a set of abstract channels to support the communication with the messages that are exchanged among the process types. These channels are part of the global state;
- a set of behavioural descriptions of the process types (they have the form of guarded events).

A control is then handled via the interaction between the components which are modelled as process types. Typically the sense/control/output steps in the standard cycle of control system are handled by the exchanges of messages through the involved components.

3.2 Event-Based Global Model: Virtual Net of Interacting Components

We define and link process types via identified common data and abstract channels for interaction (see Fig. 1). The process types are the models of the components identified within the requirements. The abstract channels are modelled according to the interaction needs. Therefore each process type uses the defined abstract channels and state, independently from the other processes; it is the *message-bus* paradigm; here the buses convey messages with data types. Note that this is a refined view which is compliant with the classical approach of modelling a distributed system by a graph whose nodes are the state machines describing processes. In our case the nodes stand for process types, and the abstract channels denote the graph edges and more specifically the interaction means: we have a virtual net of components interacting through the channels.

Therefrom, we have guidelines to help in discovering and modelling the desired behaviours of a system with various architectures, including dynamic ones. We emphasize an event-based view at global level, for composing processes. In the description of the behaviour of each process type, only the common abstract channels are used for interaction purpose, enabling thus an independent

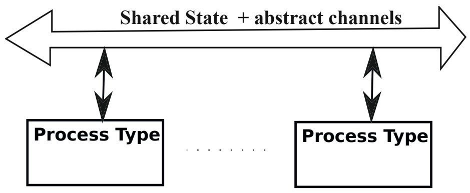


Fig. 1. Virtual net of process types

behaviour with respect to the other processes. They can be of any type, enforcing thus their heterogeneity. Consequently, the architecture of the processes which are connected via the abstract channels is highly flexible. It enables any number of process types, and any number of processes of each type to be composed. In term of distributed control, we can handle in this way the composition of any number of interacting devices and controllers.

For practical experimentation, we use the Event-B notation and method[2]. Event-B is based on first-order logic and set-theory enabling one to use the appropriate mathematical toolkit to capture modelling aspect. To introduce a few notation, total function (denoted by the symbol $\rightarrow$), surjective function (denoted by the symbol $\twoheadrightarrow$) are very useful to express easily some properties as we will see in the modelling and verification part. In the scope of the Event-B method, our process types are modelled as Event-B machines; asynchronous communication is modelled with the interleaved composition of process behaviours which are viewed as event occurrences.

Handling the Evolving of Architecture. An architecture is the set of processes of various types connected to the abstract channels at a given moment. That is the processes sharing the abstract channels conveying the message passing events from a current configuration. The configuration is submitted to restructuring or changes when the processes evolve.

An instance of a defined process type may join the configuration at any time. In the same way a process may leave a given configuration at any time. These behaviours do not change the modelling of the whole system.

Interaction Aspects. Common abstract channels are introduced to link interacting processes and make them communicate. An abstract channel is modelled as a set; we keep it abstract to handle asynchronism. But later in the specification process the channels can be refined, for example as FIFO Queues. The abstract channel is used to wait for a message or to deposit it. The interaction between the processes is then handled using the common abstract channels. Therefore, communications are achieved in a completely decoupled way to favour dynamic structuring. A process may deposit a message in the channel, generating thus an event; other processes may retrieve the message from the channel.

Therefore we use guarded events, message passing and the ordering of event occurrences; the processes synchronise and communicate through the enabling or disabling of their events. An event is used to model the waiting for a data by a process; it may be blocked until the availability of the data which enables the event guard. The availability of the data is the effect produced by another process event. Consider for example the case of processes exchanging messages, one process waits for the message, hence there should be an event with a disabled guard; another process with an enabled guard performs its behaviour which results in sending the message.

Composition of the processes. Practically the composition is implicit during the modelling of the unstructured systems considered here. But a bottom-up view may be adopted, where the composition of process types is made explicit.

The described processes P_i are combined by a *fusion* operation $\biguplus$ that merges an undefined number of process types. The semantics of the fusion operator comes from the conjunction of processes paradigms[10,1]. The fusion operator merges the state spaces and the events of the processes into a single global system Sys_g which has the conjunction of the invariants, and which in turn can also be involved in other fusion operations.

$$Sys_g \ \widehat{=}\ \biguplus_i P_i \ = \ \biguplus_i \langle S_i, E_i, Evt_i \rangle \ = \ \langle S_g, E_g, Evt_g \rangle$$

According to the fusion operator, when process types are merged, a variable denoting a set of current processes is introduced for each type; this variable is used to identify the processes of the concerned type and to distinguish the events related to each type. The processes access the global state and communicate with others, through their events.

4 Modelling and Verifying the CCTV System

From the requirements we identify the following components: cameras, screens, controllers, DVR. They have specific behaviours, they are loosely linked and their number is varying. We use abstract channels to model the shared communication links (videoChannel, $\cdots$).

The behaviour of a camera is to send a stream of captured signals to the linked screens via the control units (the controllers). The behaviour of a screen consists in visualising the streams of signals received from the cameras. A DVR saves a stream of signal from cameras and also sends them on the screens.

4.1 A Glimpse of the Constructed Model

Following our analysis, the components of the CCTV system, viewed as process types, have been gradually modelled using their weak composition. The result of the composition is a virtual net of processes which is structured using the B notation (with an abstract machine as the structuring unit). As enabled by the flexibility of the fusion operator used to weakly compose the process types, at the first abstract level, we have combined only the Cameras and the Screens in order to capture earlier the feature imposed by the requirements; it is as if the cameras are linked directly with screens. At a second abstraction level, the controllers are introduced via a refinement where the virtual net is enlarged by other processes; now it is as if the link between cameras and screens is detailed. Using this two-levels abstraction, we can handle some properties considering that the policy deployed by the controllers and hence the evolving of the architecture does not impact on the properties to be preserved. This is essentially the initial problem to be solved when considering evolving distributed system. This approach enables us to master the complexity of the model and also to master the verification of the properties. It can be used elsewhere as a modelling and verification pattern.

The structuring of the state space is achieved using identified data types, a set of state variables and an invariant that describes the properties of the processes (camera, screen):

```
MACHINE CameraHdl
...
INVARIANT     /* state space predicate */
   connectedCameras ⊆ CAMERA
      /* the set of connected cameras */
∧ connectedScreens ⊆ SCREEN
      /* the set of connected screens */
∧ activeCameras ⊆ CAMERA
∧ activeScreens ⊆ SCREEN
∧ activeCameras ⊆ connectedCameras
   /* the active cameras are part of the connected ones */
∧ activeScreens ⊆ connectedScreens
∧ display ∈ activeCameras ↠ activeScreens
   /* the active cameras are connected to active screens */
   /* All the active screens are used */
∧ videoChan ∈ ℙ(VIDEO × CAMERA)
      /* abstract channel: set of video+cameraId */
∧ nootherPriority ∈ BOOL
∧ videoStream ∈ VIDEO ⇸ activeScreens
      /* the video are displayed on one screen */
∧ displayedVideo ∈ ℙ(VIDEO)
∧ activeDVR ⊆ DVR
      /* the set of active DVR */
∧ videoStore ∈ VIDEO ⇸ activeCameras
         /* to store the video from the cameras */
∧ ···    /* more properties are added below */
```

Fig. 2. Abstract model of the CCTV system

The evolving of the system depends on the behaviour (modelled as a set of events) that defines the involved processes. The events considered for the Camera model description are summarised in Tab. 2.

Table 2. Camera handling events

Behaviour related to Camera	
Event	Description
addCamera	a new camera is added
activateCamera	one camera is activated
sendVideo	a camera sends a video
rmvCamera	a camera is removed

Table 3. Screen handling events

Behaviour related to Screen	
Event	Description
addScreen	a screen is added
getVideo	a screen gets a video
displayVideo	a screen displays a video
rmvVideo	a screen is removed
addDVR	a DVR is added
getVideoDVR	a DVR gets a video

The B specification to manage a Camera is the abstract machine equipped with these related events (see Fig. 3). In the same way the behaviour dedicated to the screen control is depicted in Fig. 4.

4.2 Mastering the Architecture and Its Modelling

One of the advantages of the composition of the process types by the fusion operator is that an important part of the model can be incrementally analysed, by considering each process type, whatever the order. Not all the behaviours can be analysed this way due to the lack of information for the interaction between the processes; but as soon as the appropriate types are introduced the interaction analysis is achieved. From the composition point of view it is very beneficial for mastering the evolving architecture. The reconfiguration of the system architecture for example does not impact on the modelling and the control of the system. However there are some limitations to this type of composition: one can neither constrain the composition nor hide some communication channels. From the heterogeneity point of view, data abstraction and behavioural abstraction allow to consider in the same way heterogeneous components via their process types.

We do not deal with the parameterisation of the architecture in this case study.

```
MACHINE CameraHdl
SETS CAMERA, SCREEN, VIDEO
VARIABLES
   connectedCameras, activeCameras, display, videoChan, nootherPriority
   activeDVR, videoStore
INVARIANT
              /* state space predicate, as given above (Fig. 4.1) */
INITIALISATION
   connectedCameras, activeCameras, display, videoChan,
   nootherPriority, activeDVR, videoStore := ∅, ∅, ∅, ∅, ∅, ∅, ∅
EVENTS
   addCamera ≙ ···
;  activateCamera ≙ ···
;  sendVideo ≙ ···
;  rmvCamera ≙ ···
END
```

Fig. 3. Structure of the Camera abstract machine

```
MACHINE ScreenHdl
SETS SCREEN, VIDEO /* abstract sets */
VARIABLES
   connectedScreens, activeScreens, display, videoChan, nootherPriority,
   videoStream, displayedVideo, activeDVR, videoStore
INVARIANT
              /* state space predicate, as given in Fig. 4.1 */
INITIALISATION
   connectedScreens, activeScreens, display, videoChan, nootherPriority,
   videoStream, displayedVideo, activeDVR, videoStore := ∅, ∅, ∅, ∅, ∅, ∅, ∅, ∅, ∅
EVENTS
   addScreen ≙ ···
;  getVideo ≙ ···
;  displayVideo ≙ ···
;  rmvVideo ≙ ···
;  addDVR  ≙ ···
;  getVideoDVR ≙ ···
;    ···
END
```

Fig. 4. Structure of the screen abstract machine

4.3 Verifying the Properties

The requirements stipulate that the system may satisfy the properties introduced in the table Tab.1 (see Sect. 2.2). Some of them are captured without restructuring. Some others are restructured; for instance, the requirement FReq_RecDispOK is rephrased as follows: all active cameras are recorded and displayed when the DVR is activated.

Properties Descriptions
FReq_DispOK: *All the captured videos are displayed on some screens.*
FReq_RecDispOK: *When the DVR is installed, all the displayed video are recorded and save to ensure the DCA.*
FReq_RecOK: *All active camera are recorded and displayed*

We have completely modelled and analyzed the system using the presented approach, including the properties formalized (in Event-B) as follows.

The property FReq_DispOK is modelled as follows:

$$\begin{array}{l}((activeDVR \neq \{\} \wedge videoStore \neq \{\}) \Rightarrow \\ \quad ((\mathsf{dom}(videoStore) \subseteq displayedVideo) \\ \qquad \vee\ (\mathsf{dom}(videoStore) \setminus \\ \qquad\quad (displayedVideo \cap \mathsf{dom}(videoStore)) \\ \qquad\qquad \subseteq \mathsf{dom}(videoStream))))\end{array}$$

The set inclusion is used in this formalisation to capture the FReq_DispOK property.

The properties FReq_cam1Ctrl and FReq_DispOK are captured through a total surjective function: $display \in activeCameras \twoheadrightarrow activeScreens$.
Indeed the domain ($activeCameras$) of the total function indicates that all the active cameras are displayed. The property FReq_RecOK is captured in the invariant with the same total surjective function (because all the active cameras that are displayed on the screens are recorded when the VCR is used).

The property FReq_NewDev is handled through the events (*addCamera*, *activateCamera*) of the machine *CameraHdl* and the event *addScreen* of the *ScreenHdl* machine; they impact on the variable *display*.

The abstract machine corresponding to the composition of the process types is obtained by the merging of the parts of the machines of the processes. The resulting machine is named IntergratedVideoSys. This later is refined in the following.

Refinement and Verification. The properties FReq_ctrlNCam and FReq_cam1Ctrl are captured through a refinement (named IntergratedVideoSys_r1) of the abstract machine modelled previously.

To master the modelling and the verification of the required properties we use the refinement technique available in the Event-B method. During the analysis of the case, one can note that the three properties FReq_DispOk, FReq_ctrlNCam and FReq_cam1Ctrl are dependent and we model them gradually. Indeed from an abstract point of view the FReq_DispOk property is captured with a single total function *display*, expressing that each camera (video) is displayed on one screen and all the videos are displayed. Thus in the first abstract machine we do not introduce the properties FReq_ctrlNCam and FReq_cam1Ctrl; they are introduced in a refinement.

The refinement we have used is summarized as follows: a function $f : A \twoheadrightarrow B$ is refined by two relations g and h defined using the same sets A, B together with a new set I such that:

$$\begin{array}{l} g : A \rightarrow I \ \wedge \ h : I \twoheadrightarrow B \wedge \\ f \subseteq (g;\ h) \end{array}$$

More specifically, I stands for the set of Controllers which have been introduced in the refinement; the total function g and the relation h are used to capture respectively the properties FReq_cam1Ctrl and FReq_ctrlNCam.

Consequently, the three properties are proved using the refinement of the previous abstract machine. This refinement process is a practical modelling and verification pattern easily reusable for similar cases of control involving controllers and controlled units in this decentralized way.

The abstract machine and its refinement have been implemented using the Rodin toolkit (see Fig. 5 for the synthetic architecture).

The proof statistics (with the refinement related to the properties FReq_DispOk, FReq_ctrlNCam and FReq_cam1Ctrl) obtained from the Rodin toolkit are drawn in the following table Tab. 4 . The context machines have their names suffixed by `Ctx`, they contain the definitions of sets which are shared by the other machines of the B project. The context machines do not generate proof obligations.

Table 4. Proof statistics

ElementName	Total PO	Auto proved	Undischarged PO
IVS_CCTV (project)	30	30	0
IntegratedVideoSysRef_Ctx	0	0	0
IntegratedVideoSys_Ctx	0	0	0
IntegratedVideoSys	24	24	0
IntegratedVideoSys_r1	6	6	0

All the proof obligations for the correctness of our models (containing the required properties) have been automatically discharged by the Rodin toolkit.

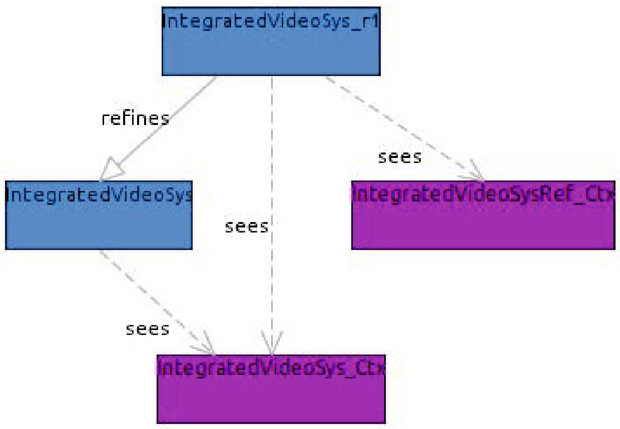


Fig. 5. Structure of the models (a snapshot from the Rodin toolkit)

4.4 Further with the Interoperability

A key to handle heterogeneity and semantic interoperability is the use of a layered structure composed of formal models, where the inner layer, the most abstract one, is the most commonly homogeneous in terms of concepts, relations and properties. The outermost layers are those with more specific details in the models. Note that the proposed method should be viewed from the abstract layer which is one of the many levels needed to master heterogeneity and semantic interoperability. Indeed, from a low abstraction level, several formal descriptions with various semantic models may be associated to one given component of a system. However changing the abstraction level to the higher one, details are forgotten until one can reach an abstract level where the semantics models are interoperable; this corresponds to the event level and the virtual component net level adopted in the current proposed work. The global properties and their analysis are only possible at this level.

Establishing bridges between formal models, using for instance the matching between domain specific ontological concepts is necessary to deal with intermediary abstraction layers. A reference compatibility layer is required as the commonly shared semantics; this is the smallest set of properties shared by all components of a system. As far as the implementation is concerned, this reference compatibility layer is captured by the invariant of our Event-B model. For this purpose, the choice of Event-B is worthwhile since Logics and Set Theory have very basic concepts which can be easily shared or implemented with other formalisms. Likewise, when we have to consider the composition of components modelled with different formalisms, on the one hand the bridging between the formalisms and on the other hand the reference to a compatibility layer are the key solutions.

Therefore a tool specific analysis is required when different formal models are considered to tackle different facets of a system. When it is necessary and possible, equivalence proofs should be conducted but this is not required for all the different facets. Interfaces between the models have to be built even with the vision of narrowing or widening the models and their coverage.

In approaches such as Ptolemy [7,8] or ModelHex [4], the compatibility between computation models or their synchronisation are emphasized; this is like a semantic adaptation in order to make the models compatible. The main difference between these approaches and our is that we do not achieve a semantic adaptation since we consider that the heterogeneity is inherently a feature of complex systems. Rather we try to handle heterogeneity but by maintaining consistency between the involved components.

5 Conclusion

We have presented a method to model and analyse a distributed control system with a varying architecture. The method is based on the composition of the identified physical and logical components of the system described at the abstract level. The components are described as process types. Their composition is based on the sharing of abstract channels denoting message buses, used by the processes to communicate without the identification of the interacting peers. This enables us to handle the distributed and dynamic architecture of the control system. The method helps to structure and model interacting components as process types. To put into practice, the Event-B notation and method are used. It follows that the refinement technique permits to handle some properties via refinement of abstract structure defined at earlier steps. Especially, in order to handle the independence of the control with respect to the controllers and their architecture, the desired relationships between the controlled units are established at abstract level as if they are directly linked; then controllers are introduced in a refinement and appear between the linked controlled units. All the properties are proved either at abstract level or in the refinement. Experimentations are conducted with a CCTV case study using AtelierB and Rodin. Our experiment is easily reusable in other case studies involving a control system with a dynamic architecture. The two-levels specification can be considered as the reference pattern when we distinguish firstly at the abstract level the end-point relationship between the controlled units (ie of the desired control policy) and secondly the structuring of the controllers and their link as intermediary agents between the controlled units. Further works are scheduled on the parametrisation of the architecture of the controllers. In the current case of the CCTV the structuring constraints are fixed. However, in some control systems for instance in embedded control systems, in order to manage energy consumption, the structure of the system can be reconfigured. This leads us to think about various behavioural modes and to define the structuring and thus the architecture with respect to these modes. The desired properties and hence their proofs will depend on the behavioural modes. But this can be as well analysed as various refinements of the same abstract machine of the ongoing architecture.

References

1. Abadi, M., Lamport, L.: Conjoining Specifications. ACM Trans. Program. Lang. Syst. 17(3), 507–535 (1995)
2. Abrial, J.-R.: Modeling in Event-B: System and Software Engineering. Cambridge University Press (2010)
3. Attiogbé, C.: Event-Based Approach to Modeling Dynamic Architecture: Application to Mobile Adhoc Network. In: Margaria, T., Steffen, B. (eds.) ISoLA 2008. CCIS, vol. 17, pp. 769–781. Springer, Heidelberg (2008)
4. Boulanger, F., Jacquet, C., Hardebolle, C., Dogui, A.: Heterogeneous model composition in modhel'x: the power window case study. In: Proceedings of Gemoc 2013, Workshop on the Globalization of Modeling Languages, Miami, Florida, USA, 10 pages (September 2013)
5. Lamport, L.: The implementation of reliable distributed multiprocess systems. Computer Networks 2, 95–114 (1978)
6. Lamport, L.: Time, Clocks, and the Ordering of Events in a Distributed System. Commun. ACM 21(7), 558–565 (1978)
7. Lee, E.A.: Disciplined Heterogeneous Modeling. In: Petriu, D.C., Rouquette, N., Haugen, Ø. (eds.) MODELS 2010, Part II. LNCS, vol. 6395, pp. 273–287. Springer, Heidelberg (2010)
8. Ptolemaeus, C. (ed.): System Design, Modeling, and Simulation using Ptolemy II. Ptolemy.org (2014)
9. Tanenbaum, A.S., van Renesse, R.: Distributed Operating Systems. ACM Comput. Surv. 17(4), 419–470 (1985)
10. Zave, P., Jackson, M.: Conjunction as Composition. ACM Transactions on Software Engineering and Methodology 2(4), 379–411 (1993)

Requirements Driven Data Warehouse Design: We Can Go Further

Selma Khouri[1,2], Ladjel Bellatreche[1], Stéphane Jean[1], and Yamine Ait-Ameur[3]

[1] LIAS/ISAE-ENSMA – Poitiers University, France
{selma.khouri,bellatreche,jean}@ensma.fr
[2] National High School for Computer Science (ESI), Algiers, Algeria
s_khouri@esi.dz
[3] ENSEEIHT/IRIT, Toulouse, France
yamine@enseeiht.fr

Abstract. Data warehouses ($\mathcal{DW}$) are defined as data integration systems constructed from a set of heterogeneous sources and user's requirements. Heterogeneity is due to syntactic and semantic conflicts occurring between used concepts. Existing $\mathcal{DW}$ design methods associate heterogeneity only to data sources. We claim in this paper that heterogeneity is also associated to users' requirements. Actually, requirements are collected from heterogeneous target users, which can cause semantic conflicts between concepts expressed. Besides, requirements can be analyzed by heterogeneous designers having different design skills, which can cause formalism heterogeneity. Integration is the process that manages heterogeneity in $\mathcal{DW}$ design. Ontologies are recognized as the key solution for ensuring an automatic integration process. We propose to extend the use of ontologies to resolve conflicts between requirements. A pivot model is proposed for integrating requirements schemas expressed in different formalisms. A $\mathcal{DW}$ design method is proposed for providing the target $\mathcal{DW}$ schema (star or snowflake schema) that meets a uniformed and consistent set of requirements.

Keywords: Data warehouse, semantic heterogeneity, formalism heterogeneity, integration, ontology-based design.

1 Introduction

Data warehouses are defined as data integration systems which data are extracted from a set of heterogeneous sources and materialized in a unified view, in order to answer business and analysis requirements collected from users and decision makers. Concept heterogeneity is one of the most critical issues in $\mathcal{DW}$ system design. Heterogeneity is often associated to data. It is caused by syntactic and semantic conflicts occurring between data stored in different sources. However, same conflicts can occur when gathering requirements from users and decision makers. For example *naming conflicts* occur when concept naming differs between users (eg. synonymy and homonymy conflicts). *Scaling conflicts* arise when different value measures are used when expressing requirements (for example the price of a product can be given in dollar or in euro). *Confounding conflicts* occur when concepts used by users seem to have the same meaning, but differ in reality due to different measuring contexts. For example, a property price is applied only to new products for user 1, but is applied to all products for user 2. *Representation conflicts* arise when designers describe the same concept in different ways. For example, customer's name is represented by two attributes *FirstName and LastName* for designer 1, and only by one attribute *Name* for Designer 2. The projection of requirements on the ontology helps to identify these conflicts and inconsistencies in order to resolve them.

A second type of heterogeneity concerns formalisms used to define the requirements model. Different formalisms can be used by designers for defining user's requirements. This situation is particularly observed with the development of global enterprises having various corporations that can spread over different states, countries or even continents, where the number of designers may increase and become strongly heterogeneous. Each designer has its own design habits. Consequently, they may use different vocabularies and formalisms to represent their requirements. This brings several challenging issues related to requirements integration: (i) how to integrate vocabularies, (ii) how to integrate the formalisms and (iii) how to identify conflicts and inconsistencies between requirements in an efficient way.

Unfortunately, most $\mathcal{DW}$ design methods focus on data integration and omit requirements integration. This can be explained by the slow evolution of $\mathcal{DW}$ design methods. In the *first generation* of $\mathcal{DW}$ design, dedicated studies have mainly concerned three phases [9]: logical design phase that defines a unified view of data, the ETL (Extract-Transform-Load) phase that extracts data from sources, transforms data if necessary and loads data into the target schema, the physical design phase that implements the final $\mathcal{DW}$ schema and defines some relevant optimization structures. Issues related to integration were managed in the ETL phase. In the *second generation* of $\mathcal{DW}$ design, two additional phases were added: requirements definition and conceptual design phase. Kimball's studies [12] introduced requirements definition in $\mathcal{DW}$ design. Different requirements driven design methods followed proposing to define $\mathcal{DW}$s from a set of users' requirements. Other hybrid methods proposed to define $\mathcal{DW}$s from both sources and requirements. The conceptual design phase has completed the design cycle in order to provide a $\mathcal{DW}$ schema independent of all implementation issues, which facilitates its validation by users However, integration issues were not studied for these additional phases, and concerned exclusively managing conflicts occurring between data sources. There is now a consensus on a typical $\mathcal{DW}$ life cycle that includes the following phases [8]: requirements definition, conceptual design, logical design, ETL phase and physical design.

Several studies were proposed in the literature related to the problem of data integration. The major progress toward automatic integration resulted from some levels of explicit representation of data meaning through ontologies [4]. Ontologies are defined as consensual and explicit representations of conceptualization. Some $\mathcal{DW}$ design methods have proposed the use of domain ontologies in order to manage conflicts between sources. The main proposition of this paper is to extend the use of ontologies in order to manage requirements conflicts. Furthermore, we take advantage of an important ontological skill: *reasoning* in order to identify different relationships between requirements, and to obtain a design schema of better quality. Three main contributions are proposed in this paper:

– Concerning formalism heterogeneity: two feasible scenarios may be offered to designers: (i) they use a generic formalism (pivot model) to express their requirements. Pivot model is a solution that reduces the complexity of exchanging different models. (ii) Designers keep using their favorite formalism and a mapping between their model and the pivot one is established. This scenario is better than the first one since it provides more autonomy to designers. We propose as a first contribution, a pivot model between three requirements formalisms usually used in $\mathcal{DW}$ design (process driven formalism, use case formalism and goal formalism).
– Concerning vocabulary heterogeneity: conflicts occurring between data stored in sources or between requirements collected from users can be solved if the meaning of each object (term) used is defined precisely and explicitly. We thus propose the use of a shared ontology integrating data sources. The second contribution consists of connecting the requirements pivot model to this ontology in order to eliminate all semantic conflicts, and unify the heterogeneous vocabularies used during requirements collection.
– Reasoning: Ontological reasoning mechanisms are then used to identify different relationships between requirements, which constitutes the third contribution. Only a set of consistent requirements is recorded to identify the target $\mathcal{DW}$ schema.

In order to realize these contributions, we propose a design method for $\mathcal{DW}$ schema definition, covering the following phases: requirements definition and analysis, conceptual design, logical design and physical design. This method takes as inputs: a shared ontology integrating sources and a set of users' requirements (collected from heterogeneous users and defined using different formalisms). It provides a $\mathcal{DW}$ schema (star or snowflake) covering a set of consistent requirements. We used *Lehigh University Benchmark*[1] (LUBM) ontology to illustrate our proposal. Figure 1 illustrates the proposed approach.

The rest of the paper is organized as follows: section 2 presents related works. Section 3 presents the proposed design method. The pivot and the ontology models are first described. Ontological reasoning mechanisms are used in order to analyze the given set of requirements. The four design phases are then presented. Section 4 presents the implementation of our approach. We illustrate how the analysis of requirements allows obtaining a $\mathcal{DW}$ schema of better quality. Section 5 concludes the paper.

[1] `swat.cse.lehigh.edu/projects/lubm/`

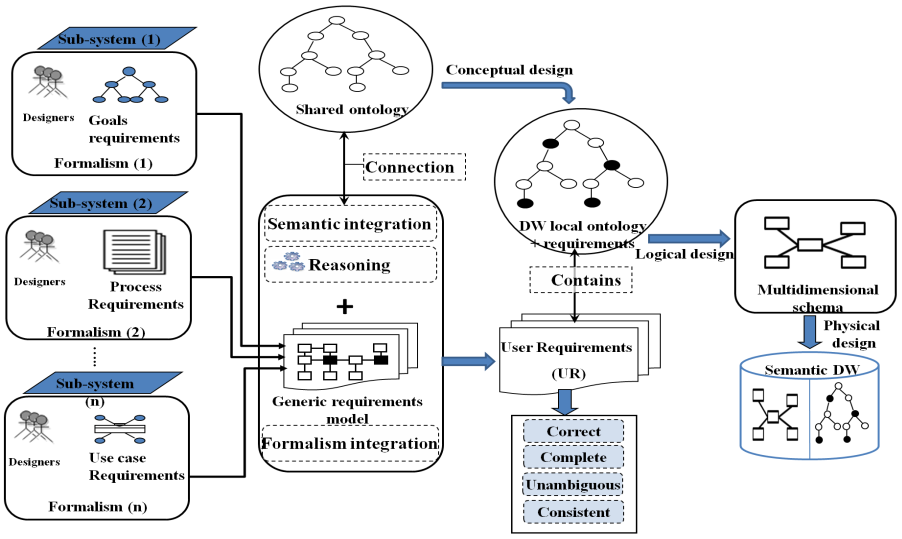


Fig. 1. Approach overview

2 Related Work

This section presents different studies related to $\mathcal{DW}$ design, we focus on efforts proposing requirements driven approaches. Then, we present studies using ontologies to define a $\mathcal{DW}$ schema. Finally, we present studies using ontologies for analyzing requirements and reasoning on them.

2.1 $\mathcal{DW}$ Design: A Requirements Driven Perspective

The purpose of $\mathcal{DW}$ design schema is to define a target schema providing a unified view of data and answering a set of requirements. This schema must handle multidimensional concepts (facts, dimension, measures, dimensions attributes and hierarchies). The $\mathcal{DW}$ schema can be defined at different abstraction levels: conceptual, logical or physical. Different methods have been proposed to define this design schema. The instability of $\mathcal{DW}$ life cycle makes most research efforts concentrate on one or two design phases. Proposed studies usually deal with the definition of $\mathcal{DW}$ schema or the ETL phase populating the schema. The ETL phase is out of the scope of this study. The definition of the requirements phase in $\mathcal{DW}$ design emerged from different studies proposing: *supply* driven, *demand* driven and *hybrid* approaches.

Requirements definition plays a crucial role in $\mathcal{DW}$ design and determines its functional behavior and all needed enterprise information. The requirement engineering process can be divided into four activities: requirements elicitation and analysis, specification, validation and management. Requirements elicitation and analysis in $\mathcal{DW}$ design literature differ according to the object analyzed.

We distinguish: (1) *Process* driven analysis [22] that analyzes requirements by identifying business processes of the organization, (2) *User* driven analysis that identifies requirements of target users and unifies them in a global model like [3,14] that develops use case models to define $\mathcal{DW}$ requirements, and (3) *Goal* driven analysis [6] that identifies goals and objectives that guide decisions of the organization at different levels. These requirements can be functional or non functional.

Lopez et al. [15] classify requirements specification techniques into three categories: (i) *informal* techniques using natural language, sometimes with structuring rules, (ii) *semi-formal* techniques generally based on graphic notations with a specified syntax like IStar or UML diagrams [14] and (iii) *formal* techniques based on mathematical or logical notations providing a precise and non-ambiguous framework for requirements modeling. For example, [11] propose to use description logic formalism to define requirements.

Several research efforts were proposed to deal with formalism heterogeneity problem. The work of [21] is an example of these studies, where the authors propose solutions to integrate semi-formal formalisms (that use diagram and tabular techniques) and formal formalisms (that use mathematics, logic or algebra). However, this effort has not been made for $\mathcal{DW}$ design.

2.2 Ontologies for Designing $\mathcal{DW}$s

Ontologies have been introduced in $\mathcal{DW}$ design for integrating heterogeneous sources. In these studies, a domain ontology is assumed existent. The set of sources reference this ontology. These references can be defined a priori during the source design, or a posteriori using matching algorithms that align sources to the ontology. Bellatreche et al. [1] provide an overview of different integration scenarios based on ontologies. Ontological methods for designing $\mathcal{DW}$s emerged recently, following both supply driven and demand driven approaches. The first two methods are mainly supply-driven, where a domain ontology is used as a schema integrating data sources: [17] defines the $\mathcal{DW}$ multidimensional model (facts and dimensions) from an OWL ontology by identifying functional dependencies (*Functional ObjectProperties*) between ontological concepts. Nebot et al. [16] define a semi-automatic method to build multidimensional tables from semantic data guided by the user requirements. We proposed in [11] a hybrid design method that extends the use of ontologies for resolving two issues: integration of sources and for the specification of the requirements model. However, formalism heterogeneity issue is not studied in this work. Romero et al. proposed in [18] a hybrid method producing a multidimensional model from an OWL ontology describing sources. Requirements are then used to identify the ETL operations needed for mapping sources to target data stores.

2.3 Ontologies for Requirements Engineering

Requirements engineering field has used ontologies since the 80's and still in recent works to support analysis and reasoning on requirements. Proposed studies provide solutions dedicated for transactional systems (not decisional ones). As instance, [10] proposed an ontological method for analyzing requirements, where a mapping between specified requirements and ontological elements is established. This ontology consists of a thesaurus and inference rules. [13] proposed an approach to improve requirements specified in natural language by the use of linguistic ontologies. [19] studied the problem of requirements expression and their refinement. The authors propose the use of goal-oriented analysis language to describe each requirement that can be refined into sub-goals. The majority of these studies manage heterogeneity of vocabularies, but they ignore the heterogeneity of the used modeling languages.

Other studies used ontologies for reasoning about requirements. As instance, Siegemund et al.[20] used ontologies for structuring concepts, requirements and relationships captured during requirements elicitation. The approach provides : an ontology-based requirements *meta model* describing meta data and requirements relationships, and a set of *consistency and completeness rules* for validating the requirement Specification. Goknil et al.[7] propose a metamodel supporting the common concepts of some requirements modeling approaches. Four types of requirements relationships are identified: Refines, Requires, Conflicts, and Contains. Based on this formalization, analysts can perform reasoning on requirements to detect implicit relations and inconsistencies. The entered requirements and their relations are stored in an OWL ontology.

We notice however that these studies are dedicated for transactional requirements. Surprisingly, no effort has been made for exploiting ontological specification and its reasoning capabilities for analyzing $\mathcal{DW}$ user's requirements in order to enhance the $\mathcal{DW}$ schema defined. Besides, the main limitation of these ontological proposals is about the consensuality of their ontologies. The ontology presented in these studies is not consensual, it is only defined to store a set of relevant requirements. This limits designers that aim to share and exchange their models with other project groups referencing the same requirements ontologies. We assume in our approach the existence of a *consensual* ontology defined by *domain experts*.

3 Preliminaries : Ontology Formalism

OWL is the ontology definition language endorsed by the World Wide Web Consortium (W3C). OWL language is based on description logic formalism (a first order logic). DL formalism is defined as the formalism used to define logics specifically designed to represent structured knowledge and to reason upon. We used DL concepts definition to formalize the ontology model. The ontology model is formally defined as follows OM: <C, R, Ref (C), Formalism>

- *C*: denotes *Concepts* of the model (atomic concepts and concept descriptions).
- *R*: denotes *Roles* (relationships) of the model. Roles can be relationships relating concepts to other concepts, or relationships relating concepts to data-values (like Integers, Floats, etc).

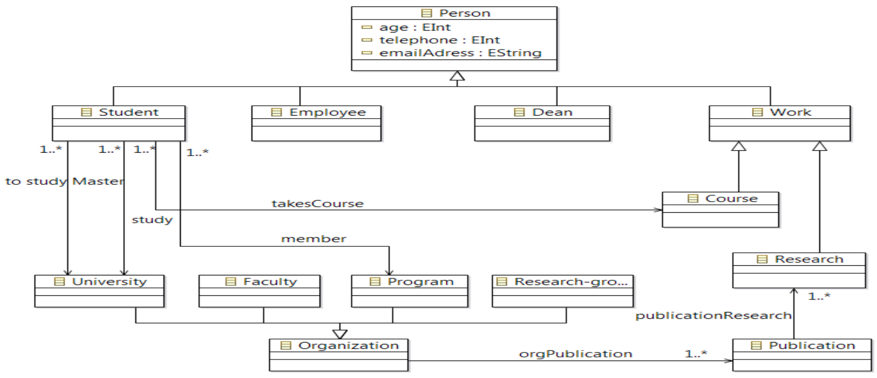


Fig. 2. A partial view of LUBM ontology schema

- $Ref : C \cup R \rightarrow$ (Operator, Exp(C,R)). Ref is a *function* defining terminological axioms of a DL TBOX. Operators can be inclusion ($\sqsubseteq$) or equality ($\equiv$). Exp(C,R) is an expression over concepts and roles of OM using constructors of description logics such as union, intersection, restriction, etc. (e.g., Ref(Student)$\rightarrow$($\sqsubseteq$, Person $\sqcap$ $\forall$takesCourse(Person, Course))).
- Formalism is the *formalism* followed by the global ontology model like RDF, OWL, etc.

In our context, we assume the existence of a shared global ontology. An ontology is shared when the sources are committed to using its ontological definitions, which are accepted and eventually standardized. Each contributor of a project shall reference that ontology "as much as possible" (i.e. each local class must reference its smallest subsuming class in the shared ontology) Locally, designers may extend it by other concepts and properties to fitful his local requirements. As consequence, each designer will have his own ontology (called local ontology). The designers may communicate through the common used concepts defined in the shared ontology.

4 Proposed Method

We present in this section the method we propose to design $\mathcal{DW}$ schemas. We describe our proposal following the design steps: requirements definition, conceptual design, logical design and physical design.

4.1 Requirements Definition

This phase includes three steps: (1) definition of the pivot model, (2) connection of the pivot model to the ontology model and (3) ontological analysis of requirements.

Definition of the Pivot Model: in order to identify the different components of our pivot model, we deeply studied three important formalisms used in requirements-driven $\mathcal{DW}$ design : Goal-Oriented formalism, Process-Oriented language (we studied MCT model, a process model of MERISE, a french modeling methodology), and UML use case formalism. We proposed in [2] the proposed pivot model. Let's take the following requirement example to illustrate the model concepts: "the system should analyze attendance of students to courses".

Three main components are identified: Actions, Results and Criteria (Fig3.(b)). Each requirement is designated by one action to accomplish (*Analyze Attendance*). If a requirement includes more than one action, it can be decomposed in multiple requirements (one for each action). Each requirements is influenced by one or many criteria (*Student*, *Course*). Each requirement have a result to fulfill (*Attendance*) that can be measured by a formal or semi formal metric (*the number of students attending the course*). Each requirement involves one or many actors that interact with the system to achieve the requirement. Two types of requirement are distinguished: *functional* and *non-functional.* Requirements can be related with each other through one of the following relationships: (*Requires*, *Refines*, *Contains* and *Conflicts*). These relationships will be populated by using reasoning rules on requirements.

Formally, we define a requirement as follows:
Requirement:$< \mathcal{A}, \mathcal{R}, \mathcal{M}, \mathcal{C} >$, in which:

- $\mathcal{A}$: the action that a system performs to yield an observable result.
- $\mathcal{R}$: the results realized by the system.
- $\mathcal{M} = \{m_1, m_2, ..., m_n\}$, a set of metrics quantifying the result.
- $\mathcal{C} = \{c_1, c_2, ..., c_n\}$, a set of sequence of criteria influencing the requirement's result.

Connection of Ontology Model to Requirements Model: the domain ontology is used in our approach as a formal and consensual domain dictionary, from which the designer can choose the most relevant concepts to express collected requirements. Requirements are structured using the proposed pivot model. They are afterwards expressed at the ontological level. In order to achieve this, we defined a mapping between coordinates of each requirement (Action and Criteria) and the resources (concepts) of the domain ontology. The connection between the ontology and requirement pivot model is presented in figure 3, where part (a) presents a fragment of the ontology metamodel connected to the pivot model (part (b)). The merged meta-model, called *OntoPivot* is defined as follows:

$\mathcal{O}ntoPivot :< \mathcal{GO}, \mathcal{P}ivot_{model} >$, Ontological Pivot, such that:

– $\mathcal{GO} : < C, R, Ref(C), Formalism >$ is the global shared ontology
– $\mathcal{P}ivot_{model}$: $< \mathcal{A}ctor, \mathcal{R}equirement, \mathcal{R}elationship >$, such that:
  - $\mathcal{R}equirement$:$< \mathcal{A}, \mathcal{R}, \mathcal{M}, \mathcal{C} >$, such that:
    - $\mathcal{A} = \{a_1, a_2, ..., a_n\}$, set of actions. For each $a \in \mathcal{A}$, $a \in 2^C U 2^R U 2^{Ref}$ (ontological domain).
    - $\mathcal{C} = \{c_1, c_2, ..., c_n\}$, set of criteria. For each $c \in \mathcal{C}$, $c \in 2^C U 2^R U 2^{Ref}$.
  - $\mathcal{R}elationships = \{Contains, Refines, Conflicts, Requires\}$, set of relations between requirements. For each $relation \in \mathcal{R}elationships$, $relation \in 2^R$.

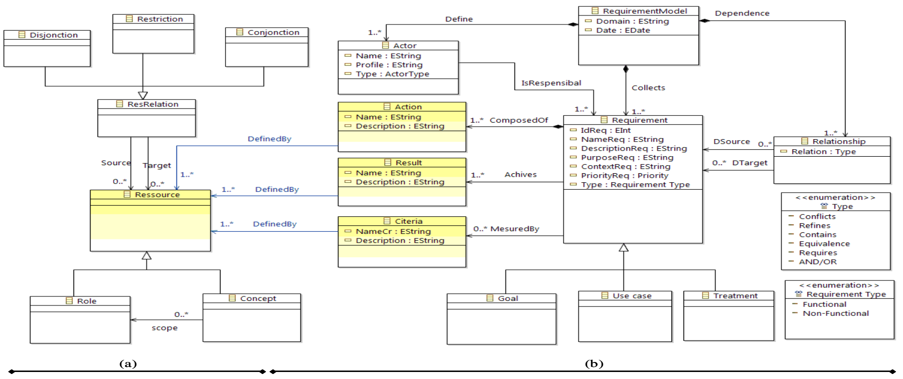


Fig. 3. Pivot metamodel connected to the ontology metamodel [2]

Note that each requirement can introduce new concepts to the ontology. For example, the requirement "the system should analyze attendance of students to courses" will be defined as a new concept in the ontology (having action, result, criteria and metric *properties*). This concept is defined as an instance of a meta concept "owl:Requirement" extending OWL meta model (see figure 4). This requirement concept can introduce a new concept "AnalyzeAttendance", whereas students and courses are already defined in the shared ontology. Consistency reasoning mechanism is used to identify incoherences and correct them. This process allows the definition of an application ontology, which combines a domain ontology and task (requirements) ontology.

Ontological Analysis of Requirements: once requirements are structured using the pivot model and expressed formally using the ontology model, they are analyzed to discover hidden relationships between them. As stated in the literature, four main relationships between requirements can be defined: contains, refines, requires and conflicts. Some reasoning mechanisms are already supported by the ontology like the equivalence between requirements concepts, others must be defined as new rules. Let's assume that: Subclass(C) is the set of subclasses of each class c $\in$ C, Role(c) is the set of roles having class c as domain, Action(R) denotes the action class of requirement R, Criteria(R) denotes the set of criteria classes of Requirement R. If a role is used to define a requirement's action, its domain class is returned. If an expression (using Ref function) is used to define requirement's action or result, it is considered as a defined class. We formally defined the following reasoning rules to identify the four relationships between requirements:

– Refinement relationship: A requirement R refines a requirement R' if R is derived from R' by adding more details to its properties [7]. Formally, refines relation is defined as follows:

  R refines R' if
  Action(R) $\sqsubseteq$ Action(R') AND
  (Criteria(R) $\subset$ Subclass(Criteria(R')) OR Criteria(R) $\subset$ Role(Criteria(R')))

  Example 1. R: The system shall analyze messages sent to individuals, teams, or all course participants at once.
  R': The system shall analyze messages sent.
  where: Action(R) and Action(R'): AnayzeMessagesSent,
  Criteria (R): {Individual, Team, Participant}, Criteria (R'): {$\top$}
  We observe that : Action(R) $\equiv$ Action(R') and Criteria(R) $\subset$ Subclass (Criteria(R'))

– Containment relationship: A requirement R contains a requirement R' if R' are parts of the whole R1 (part-whole hierarchy) [7]. Formally, containment relation is defined as follows:

  R contains R' if
  Action(R) $\sqsubseteq$ Action(R') And
  Criteria(R) $\subset$ Criteria(R')

  Example 2. R: The system shall allow lecturers to analyze enrollment policies based on grade, first-come first-serve and department.
  R': The system shall allow lecturers to analyze enrollment policies based on grade.
  where: Action(R) and Action(R'): AnalyzeEnrollment,
  Criteria (R): {Grade, Position,Department}, Criteria (R'): {Grade}
  We observe that : Action(R) $\equiv$ Action(R') and Criteria(R) $\subset$ Criteria(R')

– Conflict relationship: A requirement R conflicts with a requirement R2 if the fulfillment of R1 excludes the fulfillment of R2 and vice versa [7]. Formally, conflicts relation is defined as follows:

  R refines R' if
  Action(R) *owl : disjointWith* Action(R') And
  Criteria(R) $\subseteq$ Subclass(Criteria(R')) OR Criteria(R) $\subseteq$ Criteria(R')

  Example 3. R: The system shall allow lecturers to limit the number of students subscribing to a course.
  R': the system shall have no maximum limit on the number of course participant ever.

where: Action(R): LimitNbStudent and Action(R'): NotLimitNbStudent, Criteria (R): {Student, Course}, Criteria (R'): {Participant, Course}
We observe that : Action(R) $owl : disjointWith$ Action(R')
and Criteria(R) $\subset$ Subclass(Criteria(R'))

– Require relationship: A requirement R requires a requirement R2 if R1 is fulfilled only when R2 is fulfilled [7]. We introduce for this relation a new relation 'owl:Require' between OWL entities (E1 owl:Require E2) extending OWL meta model, that denotes that entity E1 is a precondition for entity E2. Formally, requires relation is defined as follows:

  R requires R' if
  Action(R) *owl* : *Require* Action(R') And
  Criteria(R) $\subseteq$ Subclass(Criteria(R')) OR Criteria(R) $\subseteq$ Criteria(R')

  Example 4. R: The system shall allow analyze students notification.
  R': the system shall provide messaging facilities.
  where: Action(R): AnalyzeNotification and Action(R'): ProvideMessaging, Criteria (R): {Student}, Criteria (R'): {$\top$}
  We observe that : Action(R) *owl* : *Require* Action(R')
  and Criteria(R) $\subset$ Subclass(Criteria(R'))

4.2 Conceptual Design

A $\mathcal{DW}$ ontology (DWO) viewed as a conceptual abstraction of the $\mathcal{DW}$, is defined from the global domain ontology (GO) by extracting all concepts and properties used by user requirements. Three scenarios are possible:

1. $DWO = GO$: the GO corresponds exactly to users' requirements,
2. $DWO \subset GO$: the DWO is extracted from the GO,
3. $DWO \supset GO$: the GO does not fulfill all users' requirements.

We defined in [11] different reasoning mechanisms for checking the consistency of the ontology and for identifying multidimensional concepts. We also proposed an algorithm that analyses users' requirements in order to identify the multidimensional role of concepts and properties and store them as ontological annotations. The multidimensional annotation of DWO is based on user requirement. Following Kimball's definition, we consider that each requirement Result is the fact to analyze, each of its metrics is a measure candidate for this fact, and each of its criteria is a candidate dimension. Facts are linked to dimensions by looking for one-to-many relationships between corresponding ontological concepts. Dimensions hierarchies are formed by looking for many-to-one relationships between dimensions linked to the same fact. This annotation is validated by the designer.

DWO definition extends DO formalization as follows: <C, R, Ref (C), Formalism, Multidim> where $Multidim : C \cup R \rightarrow$ Role. $Multidim$ is a function that denotes the multidimensional role (fact, dimension, measure, attribute dimension) of concepts and roles.

4.3 Logical Design

The logical model of the $\mathcal{DW}$ is generated by translating the annotated *DWO* into a relational model. Several works in the literature proposed methods for translating ontologies described in a given formalism (PLIB, OWL, RDF) to a relational or object-relational representation. This translation can follow three possible relational representations: *vertical*, *binary* and *horizontal*. Vertical representation is used for RDF ontologies, and stores data in a unique table of three columns (subject, predicate, object). In a binary representation, classes and properties are stored in tables of different structures. Horizontal representation translates each class as a table having a column for each property of the class. We proposed in [5] a set of translation rules for representing PLIB and OWL ontology (classes, properties and restrictions) in a relational schema following the binary and horizontal representations.

4.4 Physical Design

This last phase implements the final $\mathcal{DW}$ schema using a chosen DBMS. Both conventional or semantic data repositories can be used to implement the $\mathcal{DW}$ schema. Semantic data repository stores both the logical and conceptual schema (in the form of a local ontology). As we extended the ontology with the requirements model, even requirements can be stored in the repository. We implemented the obtained $\mathcal{DW}$ schema using two semantic repositories: OntoBD (academic database) and Oracle semantic database. OntoDB supports a horizontal storage layout, whereas Oracle supports a vertical storage layout.

5 Implementation

In order to implement our approach, we used LUBM ontology related to university domain, and the CMS (course management system) requirements document[2]. CMS provides a set of 60 requirements related to teaching and management of courses including interactions with students taking the course. Requirements have been adapted to a decisional application. We modified actions of requirements to analysis actions, which are more suitable for $\mathcal{DW}$ applications.

The implementation of LUBM ontology is made using *Protege* framework, defined as free, open-source ontology editor and framework for building intelligent systems (http://protege.stanford.edu/). The ontology is defined using *OWL2* language. The definition of the pivot requirements model at the ontological level is defined by the extension of OWL ontology meta model. Figure 4 illustrates this extension. Requirement meta class is defined as a new class instantiating meta class 'class', which defines all classes of the ontology. The whole pivot model is defined. Action, Criteria and Result are defined as properties of Requirement class, they have owl:Class or owl:Property as a range. Each CMS requirement is defined as an instance of this Requirement class. Relationships between requirements are defined as roles.

[2] The full requirements document is available at
`http://www.home.cs.utwente.nl/~goknila/sosym/`

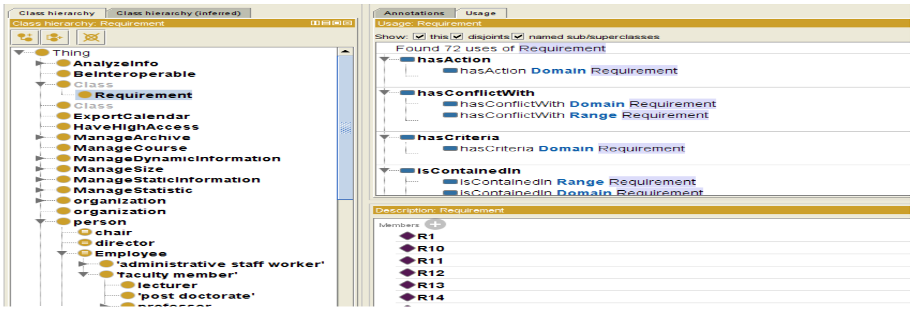


Fig. 4. OWL meta model extended with the requirement pivot model using Protege Editor

The set of reasoning rules, identifying relationships between requirements, are implemented in a java program accessing the ontology using *OWL API* (owlapi.sourceforge.net/). Each relationship inferred is stored in the ontology for the corresponding requirements. The program identified a set of relationships between defined requirements: 20 refinement relationships, 10 containment relationships, 12 require relationships and 4 conflict relationships. Figure 5 presents a set of requirements and discovered relationships between them. The schema is obtained using Protege plugin *OntoGraf*[3].

The *DWO* is defined from LUBM ontology using the modularity method *OWLExtractor*. This method is chosen because it is dedicated for OWL ontologies and it provides a Protege plugin implementing the method. The method takes as inputs the domain ontology and a signature (set of terms which will be extracted in the local ontology). In our approach, the signature corresponds to the set of requirements. We identified a subset of relevant requirements by analyzing the relationships between them. For example, refinement and containment relationships allow to eliminate some redundant requirements. When a requirement contains other requirements, the first requirement is kept, the contained requirement can be ignored. When a requirement refines another requirement, the first one gives more details (usually more criteria) to the second one. The second requirement can thus be ignored. Require relationship allow to identify the set of requirements that must be included in the final schema as they present necessary prerequisites to other requirements. Conflict relationships allow to identify requirements that cannot be fulfilled together, and cause inconsistencies. The designer must choose one of these conflictual requirements. Each requirement has a priority attribute, which can be used to eliminate requirements having the lowest priority. Require relationship can also be used. If a requirement is required by other requirements, it is more careful to not reject it.

[3] `http://protegewiki.stanford.edu/wiki/OntoGraf`

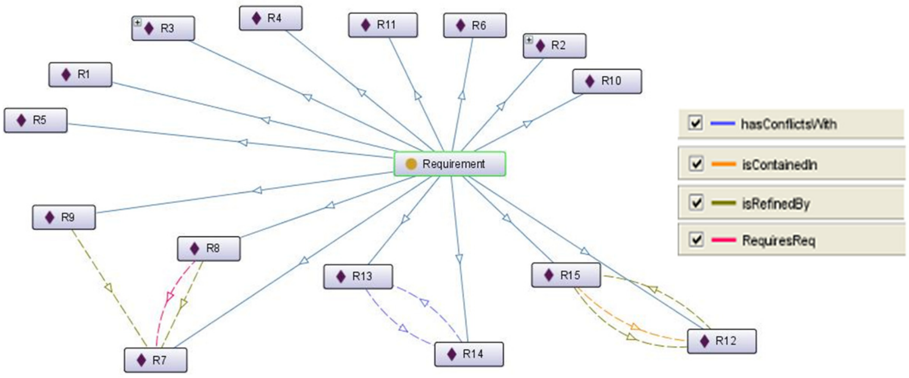


Fig. 5. Discovered relationships between requirements

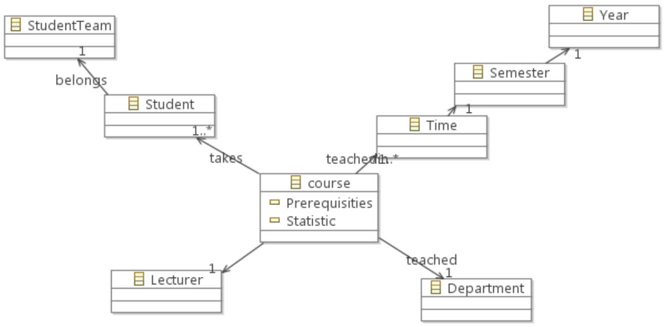


Fig. 6. $\mathcal{DW}$ Multidimensional schema obtained

The annotation algorithm is executed to annotate the extracted ontology by multidimensional annotations. Figure 6 illustrates the obtained multidimensional schema.

The reasoning rules help us to obtain a $\mathcal{DW}$ schema of better quality. Suppose that step 2 (analysis of requirements) is ignored. This would provide a schema containing conflict requirements. For example, the schema cannot answer the non functional requirement stating that the system should "limit space of storage for courses", and another requirement stating that the system should "maximize space of specific courses ". The schema would include redundant concepts due to the presence of containment and refinement relations between requirements. In fact, instead of managing and validating 60 requirements, we just have to manage 40 requirements. The validation of this schema is easier since it has to be validated by a consistent subset of requirements.

6 Conclusion

Various $\mathcal{DW}$ design methods have been proposed covering different design phases: conceptual, logical, physical and ETL design phases. Most of these methods consider integration issues related to data, but ignore requirements integration. User's requirements are collected from heterogeneous users, which usually causes semantic conflicts. Requirements are analyzed and formalized by different designers, which can cause schematic and formalisms heterogeneity. We propose in this paper to manage requirements integration for $\mathcal{DW}$ definition through an ontology-based design method. The method takes as inputs a set of requirements, and a shared ontology integrating sources. For handling formalisms heterogeneity, we defined a pivot model between three formalisms usually used for $\mathcal{DW}$ requirements models (process, use case and goal formalisms). The pivot model is connected to the shared ontology. This connection allows expressing requirements using ontological concepts which eliminates semantic conflicts. It also allows reasoning on requirements in order to identify semantic relationships between requirements (refine, contain, require and conflict relationships). The $\mathcal{DW}$ schema is then defined by following three design stages: conceptual, logical and physical design. The target $\mathcal{DW}$ schema is defined from a set of coherent and consistent requirements. We illustrated the proposed approach using LUBM ontology and requirements defined in the CMS requirements document.

There are different open issues that we are currently working on like: the management of requirements evolution, the completion of the approach with the ETL process for loading data, and the evaluation of the approach in a large scale case study in which we evaluate $\mathcal{DW}$ quality metrics and get designers feedback.

References

1. Bellatreche, L., Dung, N.X., Pierra, G., Hondjack, D.: Contribution of ontology-based data modeling to automatic integration of electronic catalogues within engineering databases. Computers in Industry 57(8), 711–724 (2006)
2. Boukhari, I., Bellatreche, L., Khouri, S.: Efficient, unified, and intelligent user requirement collection and analysis in global enterprises. In: Proceedings of International Conference on Information Integration and Web-based Applications & Services, p. 686. ACM (2013)
3. Bruckner, R., List, B., Schiefer, J.: Developing requirements for data warehouse systems with use cases. In: Proc. 7th Americas Conf. on Information Systems, pp. 329–335 (2001)
4. Doan, A., Halevy, A.Y., Ives, Z.G.: Principles of Data Integration. Morgan Kaufmann (2012)
5. Fankam, C.: OntoDB2 : Un systeme flexible et efficient de Base de Donnees á Base Ontologique pour le Web semantique et les donnees techniques. PhD thesis, ENSMA (December 2009)
6. Giorgini, P., Rizzi, S., Garzetti, M.: Goal-oriented requirement analysis for data warehouse design. In: Proceedings of the 8th ACM International Workshop on Data Warehousing and OLAP, pp. 47–56. ACM (2005)

7. Goknil, A., Kurtev, I., Berg, K., Veldhuis, J.-W.: Semantics of trace relations in requirements models for consistency checking and inferencing. Softw. Syst. Model. 10, 31–54 (2011)
8. Golfarelli, M.: From user requirements to conceptual design in data warehouse design a survey. In: Data Warehousing Design and Advanced Engineering Applications Methods for Complex Construction, pp. 1–16 (2010)
9. Inmon, W.H.: Building the data warehouse. J. Wiley (2002)
10. Kaiya, H., Saeki, M.: Ontology based requirements analysis: Lightweight semantic processing approach. In: Proceedings of the Fifth International Conference on Quality Software, pp. 223–230. IEEE Computer Society (2005)
11. Khouri, S., Boukhari, I., Bellatreche, L., Jean, S., Sardet, E., Baron, M.: Ontology-based structured web data warehouses for sustainable interoperability: Requirement modeling, design methodology and tool. Computers in Industry, 799–812 (2012)
12. Kimball, R., Reeves, L., Thornthwaite, W., Ross, M., Thornwaite, W.: The Data Warehouse Lifecycle Toolkit: Expert Methods for Designing, Developing and Deploying Data Warehouses, 1st edn. John Wiley & Sons, Inc., New York (1998)
13. Körner, J.S., Torben, B.: Natural language specification improvement with ontologies. Int. J. Semantic Computing 3, 445–470 (2009)
14. List, B., Schiefer, J., Tjoa, A.M.: Process-oriented requirement analysis supporting the data warehouse design process a use case driven approach. In: Ibrahim, M., Küng, J., Revell, N. (eds.) DEXA 2000. LNCS, vol. 1873, pp. 593–603. Springer, Heidelberg (2000)
15. López, O., Laguna, M.A., García, F.J.: Metamodeling for requirements reuse. In: Anais do WER02-Workshop em Engenharia de Requisitos, Valencia, Spain (2002)
16. Nebot, V., Berlanga, R.: Building data warehouses with semantic web data. Decision Support Systems (2011)
17. Romero, O., Abelló, A.: Automating multidimensional design from ontologies. In: Proceedings of the ACM Tenth International Workshop on Data Warehousing and OLAP, pp. 1–8. ACM (2007)
18. Romero, O., Simitsis, A., Abelló, A.: Gem: Requirement-driven generation of etl and multidimensional conceptual designs. In: Data Warehousing and Knowledge Discovery, pp. 80–95 (2011)
19. Saeki, M., Hayashi, S., Kaiya, H.: A tool for attributed goal-oriented requirements analysis. In: 24th IEEE/ACM International Conference on Automated Software Engineering, pp. 674–676 (2009)
20. Siegemund, K., Edward, J., Thomas, Y., Yuting, Z., Pan, J., Assmann, U.: Towards ontology-driven requirements engineering. In: 7th International Workshop on Semantic Web Enabled Software Engineering (October 2011)
21. Wieringa, R., Dubois, E.: Integrating semi-formal and formal software specification techniques. Information Systems 23(3-4), 159–178 (1998)
22. Winter, R., Strauch, B.: A method for demand-driven information requirements analysis in data warehousing projects. In: Proceedings of the 36th Annual Hawaii International Conference on System Sciences, 2003, pp. 9–19. IEEE (2003)

On Implicit and Explicit Semantics: Integration Issues in Proof-Based Development of Systems[⋆]

Version to Read

Yamine Ait-Ameur[1], J. Paul Gibson[2], and Dominique Méry[3]

[1] IRIT - ENSEEIHT. Institut de Recherche en Informatique de Toulouse - École Nationale Supérieure d' Électrotechnique, d'Électronique, d'Informatique, d'Hydraulique et des Télécommunications (ENSEEIHT)
`yamine@enseeiht.fr`

[2] Département Logiciels-Réseaux, IT-SudParis, Évry, France
`paul.gibson@it-sudparis.eu`

[3] Université de Lorraine, LORIA CNRS UMR 7503, Vandœuvre-lès-Nancy, France
`mery@loria.fr`

Abstract. All software systems execute within an environment or context. Reasoning about the correct behavior of such systems is a ternary relation linking the requirements, system and context models. Formal methods are concerned with providing tool (automated) support for the synthesis and analysis of such models. These methods have quite successfully focused on binary relationships, for example: validation of a formal model against an informal one, verification of one formal model against another formal model, generation of code from a design, and generation of tests from requirements. The contexts of the systems in these cases are treated as second-class citizens: in general, the modelling is implicit and usually distributed between the requirements model and the system model. This paper is concerned with the explicit modelling of contexts as first-class citizens and illustrates concepts related to implicit and explicit semantics on an example using the Event B language.

Keywords: Verification, modelling, Contexts, Domains.

1 Introduction: Implicit versus Explicit — The Need for Formality

In general usage, "explicit" means clearly expressed or readily observable whilst "implicit" means implied or expressed indirectly. However, there is some inconsistency regarding the precise meaning of these adjectives. For example, in logic and belief models [1] "a sentence is explicitly believed when it is actively held to be true by an agent and implicitly believed when it follows from what is believed."

[⋆] This work was supported by grant ANR-13-INSE-0001 (The IMPEX Project `http://impex.loria.fr`) from the Agence Nationale de la Recherche (ANR).

However, in the semantic web [2] "Semantics can be implicit, existing only in the minds of the humans [. . .]. They can also be explicit and informal, or they can be formal." The requirements engineering community use the terms to distinguish between declarative (descriptive) and operational (prescriptive) requirements [3] where they acknowledge the need for "a formal method for generating explicit, declarative, type-level requirements from operational, instance-level scenarios in which such requirements are implicit". We propose a formal treatment of the adjectives implicit and explicit when engineering *software and/or systems*.

Nowadays, several research approaches aim at formalizing mathematical theories for the formal development of systems. Usually, these theories are defined within contexts, that are imported and and/or instantiated. They usually represent the implicit semantics of the systems and are expressed by types, logics, algebras, etc. based approaches. To our knowledge, no work adequately addresses the formal description of domains expressing the semantics of the universe in which the developed systems run and their integration in the formal development process. This domain information is usually defined in an "ontology" [4].

Several relevant properties are checked by the formal methods. These properties are defined on the implicit semantics associated to the formal technique being used. When considering these properties in their context with the associated explicit semantics, these properties may be no longer respected. Without a more formal system engineering development approach, based on separation of implicit and explicit, the composition of software and/or system components in common contexts risks compromising correct operation of the resulting system. This is a significant problem when software and/or systems are constructed from heterogeneous components [5] that must be reliable in unreliable contexts [6].

To clarify, this paper is concerned with the separation of concerns when reasoning about properties of models. Although the concerns need to be cleanly separated, the models need to be tightly integrated: achieving both is a significant challenge.

2 Integrating Implicit and Explicit: Formal Methods and Ontologies

Allowing formal methods users and developers to integrate — in a flexible and modular manner — both the implicit semantics, offered by the formal method semantics, and the explicit semantics, provided by external formal knowledge models like ontologies, is a major challenge. Indeed, the formal models should be defined in the formal modelling language being used, and explicit reference and/or annotation mechanisms must be provided for associating explicit semantics to the formal modelling concepts. Once this integration is realized, the formalisation and verification of several properties related to the heterogeneous models' integration becomes possible. The most important properties that need to be addressed relate to interoperability, adaptability, dissimilarity, reconfigurability and identification of degraded modes. Refinement/instantiation and composition/decomposition could play a major role for specifying and verifying these properties. Currently, no formal method or formal technique provides explicit means for handling such an integration.

In the context of formal methods, it is well known that several formal methods for system design and verification have been proposed. These techniques are well established on a solid formal basis and their domain of efficiency, their strengths and weaknesses are well acknowledged by the formal methods community. Although, some ad-hoc formalisation of domain knowledge [7] within formal methods is possible, none of these techniques offers a built-in mechanism for handling explicit semantics.

Regarding ontologies and domain modelling, most of the work has been achieved in the large semantic web research community. There, the problem consists of annotating web pages and documents with semantic information belonging to ontologies. Thus, ontologies have mainly been used for assigning meanings and semantics to terms occurring in documents. Once, these meanings are assigned, formal reasoning can be performed due to the ontologies being based on descriptive logic. In general, however, the documents to be annotated do not conform to any model and the domain associated to the documents is not fixed. Therefore, ontologies behave like a model associated to the resources that are annotated.

We propose an integration of both worlds. On the one hand, formal methods facilitate prescriptive modelling whereas, on the other hand, ontologies provide mechanisms for explicit descriptive semantics. We conclude by noting that, in most cases, the formal models are usually defined in a fixed and limited application domain well understood by the developers.

3 A Simple Example

The illustration of the addressed problem and the underlying ideas are given in this section through a simple case study. As a first step, we demonstrate a typical development involving solely a formal model; and in a second step we show how formalized explicit knowledge contributes to identifying relevant problems related to heterogeneity.

Let us consider a simple system issued from avionic system design. We identify two sub-systems: the first one is part of the flight management system acting in the closed world (heart of the avionic systems), it produces flight information like altitude and speed; and the second is the display part of a passenger information system (open world). It displays, to the passengers, information issued from the closed world, here altitude and speed. The information is transmitted from the closed world to the open world within a communication bus. Communications are unidirectional from the closed world to the open world only.

The development of this system considers a formally expressed specification which is refined twice. Figure 1 shows the structure of the development for this case study. The next two subsections show the two proposed formal developments expressed within the Event B formal method.

We note that this example is intended only as a proof-of-concept. Its goal is not to demonstrate the power of our approach, it shows only that there is

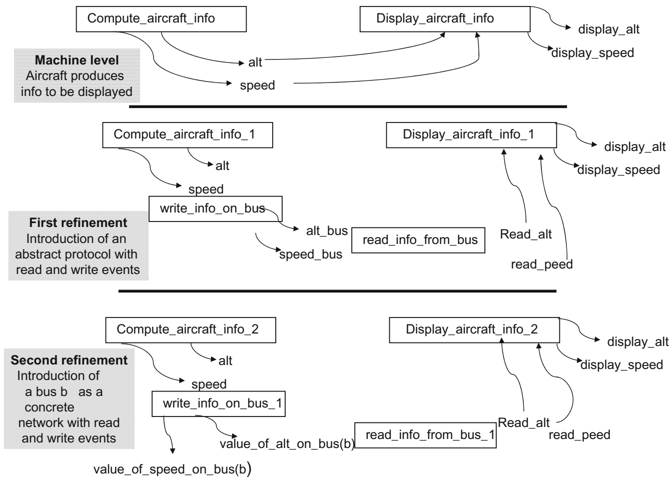


Fig. 1. A global view of the formal development

utility in separating the implicit and explicit semantics and that there is at least one such way of doing this separation in Event-B. This demonstrates that a fully formal and automated approach is feasible. Further work — on a range of case studies — will examine and compare different mechanisms for implementing the approach, with particular emphasis on scaleability and universality: can we model much larger, heterogeneous, domains of knowledge?

3.1 Formal Model with Implicit Semantics

In the implicit semantics, the models are constructed within the modelling capabilities offered by the modelling language.

The Machine Level: Specification. The first formal specification of the problem expresses that the system should communicate a computed value to be displayed. Variables, described in the invariant clauses, are *speed* (recording the effective speed), *alt* (recording the effective altitude), *display_speed* (recording the displayed speed), *display_alt* (recording the displayed altitude) and *consumed* (recording the control and the synchronisation between the produced and the consumed values. Two events are defined, one producing the information to be displayed and a second displaying this information.

```
EVENT Compute_aircraft_info
   WHEN
      grd1 : consumed = 1
   THEN
      act1 : alt :∈ ℕ1
      act2 : speed :∈ ℕ1
      act3 : consumed := 0
   END
```

```
inv1 : speed ∈ ℕ1
inv2 : alt ∈ ℕ1
inv3 : consumed ∈ {0, 1}
inv4 : display_speed ∈ ℕ1
inv5 : display_alt ∈ ℕ1
```

```
EVENT Display_aircraft_info
   WHEN
      grd1 : consumed = 0
   THEN
      act1 : display_alt := alt
      act2 : display_speed := speed
   END
```

Event Compute_aircraft_info models the update step for the altitude and the speed; the *consumed* variable is set to 0 and the event Display_aircraft_info is triggered when the variable $Display\_aircraft\_info$ is updated by the sent value.

The invariant types and constrains the variables within numerical bounds; it does not take into account the fact that the produced values and consumed values belong to different domains. The *ontological context* is to provide information for relating these two domains. The main idea is to *annotate* the model by expressing the knowledge domain using the context models in figure 2.

First Refinement: Introducing an Abstract Communication Protocol. The new model First_Refinement extends the state by new variables recording the traffic of messages through a bus. It specifies that we have to manage the transmission of messages with the addition of new control variables (*written*, *read*, *displayable*). Two new events model the reading and the writing to and from the bus. The two abstract events are refined by strengthening guards with respect to the new control variables (*read*, *written*, *displayable*). The new model introduces an abstract protocol for the bus.

```
EVENT compute_aircraft_info_1
   REFINES compute_aircraft_info
   WHEN
      grd1 : consumed = 1
      grd2 : written = 1
      grd3 : read = 1
      grd4 : displayable = 1
   THEN
      act1 : alt :∈ ℕ1
      act2 : speed :∈ ℕ1
      act3 : consumed := 0
   END
```

```
EVENT Display_aircraft_info_1
   REFINES Display_aircraft_info
   WHEN
      grd1 : consumed = 0
      grd2 : written = 0
      grd3 : read = 0
      grd4 : displayable = 1
   THEN
      act1 : display_alt := read_alt
      act2 : display_speed := read_speed
      act3 : displayable := 0
   END
```

The two next events model the abstract protocol for exchanging the data. They describe the fact that a value is written to and then read from an abstract bus.

```
EVENT write_info_on_bus
   WHEN
      grd1 : consumed = 0
      grd2 : written = 1
      grd3 : read = 1
      grd4 : displayable = 1
   THEN
      act1 : alt_bus := alt
      act2 : speed_bus := speed
      act3 : written := 0
   END
```

```
EVENT read_info_from_bus
   WHEN
      grd1 : consumed = 0
      grd2 : written = 0
      grd3 : read = 1
      grd4 : displayable = 1
   THEN
      act1 : read_alt := alt_bus
      act2 : read_speed := speed_bus
      act3 : read := 0
   END
```

Second Refinement: Concretizing the Bus for Communication. The current system is still abstract and we have to add details concerning the bus. Finally, the four events of the model First_Refinement are refined to concretize actions over the bus b. The two first events are directly related to the computation and display components.

```
EVENT compute_aircraft_info_2
  REFINES compute_aircraft_info_1
  WHEN
    $grd1 : consumed = 1$
    $grd2 : written = 1$
    $grd3 : read = 1$
    $grd4 : displayable = 1$
  THEN
    $act1 : alt :\in \mathbb{N}_1$
    $act2 : speed :\in \mathbb{N}_1$
    $act3 : consumed := 0$
  END
```

```
EVENT Display_aircraft_info_2
REFINES Display_aircraft_info_1
WHEN
  $grd1 : consumed = 0$
  $grd2 : written = 0$
  $grd3 : read = 0$
  $grd4 : displayable = 1$
THEN
  $act1 : display\_alt := read\_alt$
  $act2 : display\_speed := read\_speed$
  $act3 : displayable := 0$
END
```

The next two events are the operations over the bus. They precise how the *speed* and *altitude* information are written to and read from the bus b

```
EVENT read_info_from_bus_2
  REFINES read_info_from_bus
WHEN
  $grd1 : consumed = 0$
  $grd2 : written = 0$
  $grd3 : read = 1$
  $grd4 : displayable = 1$
  THEN
  $act1 : read\_alt := value\_of\_alt\_on\_bus(b)$
  $act2 : read\_speed := value\_of\_speed\_on\_bus(b)$
  $act3 : read := 0$
  END
```

```
EVENT write_info_on_bus_2
REFINES write_info_on_bus
WHEN
  $grd1 : consumed = 0$
  $grd2 : written = 1$
  $grd3 : read = 1$
  $grd4 : displayable = 1$
THEN
  $act1 : value\_of\_alt\_on\_bus(b) := alt$
  $act2 : value\_of\_speed\_on\_bus(b) := speed$
  $act5 : written := 0$
END
```

3.2 Formal Model with Explicit Semantics

The previous development follows the formal modelling approach provided by the Event B method, focusing on the binary relation referred to in the introduction of this paper.

The second development, presented below, introduces the explicit knowledge carried out by ontologies, it is used for coding the ternary relationship referred to in the introduction. In the case of Event B, it is formalized within *contexts.* The ternary relationship is obtained by *annotation* i.e. linking the model elements, variables in our case, to the explicit knowledge. In the following we illustrate the process of handling explicit domain knowledge in Event B models, using the same aircraft case study as before.

Contexts for Defining Explicit Domain Knowledge. The first step consists of introducing the explicit domain knowledge through a formal model for ontologies. It will be used to annotate the concepts seen in the previous models.

In the simple case we are addressing, this knowledge is defined by contexts (see figure 2). In this case, we are concerned by the description of the units that may be associated to the *altitude* and to the *speed*.

Meters, inches, kilometers per hour, and miles per hour are introduced to define distance speed measures. Conversion functions, that define equivalences

CONTEXT $domain\_knowledge\_for\_units$
CONSTANTS
$inches, meters, mph, kph, inch2meters, mphour2kphour$
AXIOMS
$axm1 : inches \subseteq \mathbb{N}_1$
$axm2 : meters \subseteq \mathbb{N}_1$
$axm3 : mph \subseteq \mathbb{N}_1$
$axm4 : kph \subseteq \mathbb{N}_1$
$axm5 : inches \neq \varnothing$
$axm6 : meters \neq \varnothing$
$axm7 : mph \neq \varnothing$
$axm8 : kph \neq \varnothing$
$axm9 : inch2meters \in inches \rightarrow meters$
$axm10 : mphour2kphour \in mph \rightarrow kph$
END

Fig. 2. The ontological context

in terms of ontology definitions, are described by the functions *inch2meters* and *mphour2kphour*. We do not detail the defintions of these two functions but they can be made more precise by an implementation step at a later phase in the process.

Annotation: Associating Explicit Knowledge to Model Variables. Once the explicit knowledge has been formalized, it becomes possible to annotate the concepts available in the obtained formal models. In our case, the variables are annotated by explicitly referring to the ontology defined in the context of figure 2. Measurement units are introduced in an explicit way.
The variables are then defined as follows. When the annotations have been specified, the verification of the previous development defined in section 3.1 is no longer correct. Some proof obligations cannot be satisfied due to incoherent assignments.

$inv1 : speed \in mph$
$inv2 : alt \in inches$
$inv3 : consumed \in \{0, 1\}$
$inv4 : display\_speed \in kph$
$inv5 : display\_alt \in meters$

The new invariant defines the ontological constraints that should be satisfied by the events. For example, one of the generated proof obligations for checking the preservation of $inv5 : display\_alt \in meters$ by the event Display_aircraft_info fails to prove that $alt \in meters$. Thus, we should modify the event Display_aircraft_info by removing the previous *act1* and *act2* and by adding the ontological information provided by the two functions *inch2meters* and *mph2kphour* in the rewritten actions *nact1* and *nact2* . The example is simple and gives an obvious way to solve the unproved proof obligation: without refinement it may be much more difficult to discover why similar proof obligations are not discharged.

Consequently, the following events — Display_air craft_info and Compute_aircraft _info — require further description. In Particular, Display_aircraft_info has been modified in order to handle converted values issued from Compute_aircraft_info.

```
EVENT Compute_aircraft_info
WHEN
   $grd1 : consumed = 1$
   THEN
      $act1alt :\in inches$
      $act2speed :\in mph$
      $act3consumed := 0$
END
```

```
EVENT Display_aircraft_info
   WHEN
      $grd1 : consumed = 0$
   THEN
      $nact1display\_alt := inch2meters(alt)$
      $nact2 : display\_speed := mphour2kphour(speed)$
   END
```

First Refinement: Introducing an Abstract Communication Protocol. As a next step, we can add new features in the current model Main_exchange by refining it into First_Refinement_Dom.

The new model First_Refinement_Dom performs the same extension of the state as in the previous case using implicit knowledge. This is quite natural since none of these state variables (i.e. *written*, *read*, *displayable*) are annotated. Two new events model the reading to and the writing from the bus. The invariant is extended by sub-invariants $inv6 \ldots inv15$. Notice the introduction of new kinds of invariants, labeled $inv13$ and $inv14$, borrowed from the context where the explicit knowledge is described. They define ontological invariants.

```
$inv1\ to\ inv5\ of\ last\ model$
$inv6 : speed\_bus \in kph$
$inv7 : read\_alt \in meters$
$inv8 : read\_speed \in kph$
$inv9 : alt\_bus \in meters$
$inv10 : written \in \{0, 1\}$
$inv11 : read \in \{0, 1\}$
$inv12 : displayable \in \{0, 1\}$
$inv13 : (written = 0) \Rightarrow (alt\_bus = inch2meters(alt) \wedge speed\_bus = mphour2kphour(speed))$
$inv14 : (read = 0) \Rightarrow (read\_alt = alt\_bus \wedge read\_speed = speed\_bus)$
$inv15 : (displayable = 0) \Rightarrow (display\_alt = read\_alt \wedge display\_speed = read\_speed)$
```

The two abstract events are refined by strengthening guards with respect to the new control variables (*read*, *written*, *displayable*). The new model introduces an abstract protocol for the bus.

```
EVENT compute_aircraft_info_1
   REFINES compute_aircraft_info
   WHEN
      $grd1 : consumed = 1$
      $grd2 : written = 1$
      $grd3 : read = 1$
      $grd4 : displayable = 1$
   THEN
      $act1 : alt :\in inches$
      $act2 : speed :\in mph$
      $act3 : consumed := 0$
   END
```

```
EVENT Display_aircraft_info_1
   REFINES Display_aircraft_info
   WHEN
      $grd1 : consumed = 0$
      $grd2 : written = 0$
      $grd3 : read = 0$
      $grd4 : displayable = 1$
   THEN
      $act1 : display\_alt := read\_alt$
      $act2 : display\_speed := read\_speed$
      $act3 : displayable := 0$
   END
```

The two next events model the abstract protocol for exchanging the data. The abstract protocol manages the relationship between the measurement units. The ontological annotation appears in the invariant $inv13$: the protocol ensures the *correct* communication.

```
EVENT write_info_on_bus
  WHEN
    $grd1 : consumed = 0$
    $grd2 : written = 1$
    $grd3 : read = 1$
    $grd4 : displayable = 1$
  THEN
    $act1 : alt\_bus := inch2meters(alt)$
    $act2 : speed\_bus := mphour2kphour(speed)$
    $act3 : written := 0$
  END
```

```
EVENT read_info_from_bus
  WHEN
    $grd1 : consumed = 0$
    $grd2 : written = 0$
    $grd3 : read = 1$
    $grd4 : displayable = 1$
  THEN
    $act1 : read\_alt := alt\_bus$
    $act2 : read\_speed := speed\_bus$
    $act3 : read := 0$
  END
```

Context Extension: Need of Explicit Knowledge for the Bus. The current system is still abstract and we have to add details concerning the bus. Following good engineering practice, the communication bus should be described independently of any usage in a given model. Here again, an ontology of communication medias is needed. It is defined in a context that extends the one defined for measure units. The bus has specific properties that are expressed in a new context domain_knowledge_for_protocols(in figure 3.2).

```
CONTEXT $domain\_knowledge\_for\_protocols$
EXTENDS $domain\_knowledge\_for\_units$
SETS
  $bus, bus\_type$
CONSTANTS
  $unidirectional, bidirectional, type\_of\_bus$
AXIOMS
  $axm1 : bus\_type = \{unidirectional, bidirectional\}$
  $axm2 : type\_of\_bus \in bus \rightarrow bus\_type$
  $axm3 : bus \neq \varnothing$
  $axm4 : \exists bb \cdot (bb \in bus \land type\_of\_bus(bb) = unidirectional)$
END
```

Fig.3.Context for the bus

Notice that the definition of explicit knowledge is modular. It uses contexts that import only those ontologies that are needed for a given development. Moreover, it is flexible since contexts can be changed, if the domain knowledge or the nature of the manipulated concepts evolves. The whole formal development of the system does not need to be rewritten.

Second Refinement: Concretizing the Bus for Communication. The new invariant extends the previous one, whilst integrating the state of the bus. It also asserts that the bus is unidirectional. The invariant $Ninv3 : b \in bus$ is an ontological invariant and the context enriches the description of the domain.

```
$Ninv1 : value\_of\_speed\_on\_bus \in bus \rightarrow kph$
$Ninv2 : value\_of\_alt\_on\_bus \in bus \rightarrow meters$
$Ninv3 : b \in bus$
$Ninv4 : (written = 0) \Rightarrow (value\_of\_speed\_on\_bus(b) = mphour2kphour(speed))$
$Ninv5 : (written = 0) \Rightarrow (value\_of\_alt\_on\_bus(b) = inch2meters(alt))$
$Ninv51 : type\_of\_bus(b) = unidirectional$
$Ninv6 : (read = 0) \Rightarrow (read\_speed = value\_of\_speed\_on\_bus(b))$
$Ninv7 : (read = 0) \Rightarrow (read\_alt = value\_of\_alt\_on\_bus(b))$
$Ninv8 : alt\_bus = value\_of\_alt\_on\_bus(b)$
$Ninv9 : speed\_bus = value\_of\_speed\_on\_bus(b)$
```

Finally, the four events of the model First_Refinement_Dom are refined to concretize the actions over the bus b. The two first events are directly related to the computation and display components.

```
EVENT compute_aircraft_info_2
REFINES compute_aircraft_info_1
  WHEN
    grd1 : consumed = 1
    grd2 : written = 1
    grd3 : read = 1
    grd4 : displayable = 1
  THEN
    act1 : alt :∈ inches
    act2 : speed :∈ mph
    act3 : consumed := 0
  END
```

```
EVENT Display_aircraft_info_2
REFINES Display_aircraft_info_1
WHEN
  grd1 : consumed = 0
  grd2 : written = 0
  grd3 : read = 0
  grd4 : displayable = 1
THEN
  act1 : display_alt := read_alt
  act2 : display_speed := read_speed
  act3 : displayable := 0
END
```

The two next events — read_info_from_bus_2 and write_info_on_bus_2 — model operations over the bus. They both deal with ontological annotations, where the more detailed characteristics of the bus are necessary for guaranteeing the safety of the global system.

```
EVENT read_info_from_bus_2
  REFINES read_info_from_bus
  WHEN
    grd1 : consumed = 0
    grd2 : written = 0
    grd3 : read = 1
    grd4 : displayable = 1
  THEN
    act1 : read_alt := value_of_alt_on_bus(b)
    act2 : ( read_speed :=
             value_of_speed_on_bus(b) )
    act3 : read := 0
  END
```

```
EVENT write_info_on_bus_2
REFINES write_info_on_bus
  WHEN
    grd1 : consumed = 0
    grd2 : written = 1
    grd3 : read = 1
    grd4 : displayable = 1
  THEN
    act1 : ( value_of_alt_on_bus(b) :=
             inch2meters(alt) )
    act2 : ( value_of_speed_on_bus(b) :=
             mphour2kphour(speed) )
    act5 : written := 0
  END
```

The summary of proof obligations tells us that the proof is not complex. In fact, the example is simple and does not require further interaction as the ontological annotations help to automatically derive the proofs.

4 Discussion

4.1 Proof-Based Development Methods for Safe and Secure Models and Systems: The Importance of Refinement

Deductive verification for program correctness has outstanding challenges. Formal methods toolsets assist the developer who is trying to check a set of proof obligations using a proof assistant. In contrast to these semi-automatic proof techniques, model checking [8] appears to be a better solution when the developer does not want to interact with the proof tool and, although model checking is addressing specific systems with a reasonable size or is applied on abstractions of systems to facilitate the proof, there are limits to the use of model checking-based techniques. Finally, another solution is to play with abstractions and to apply the abstract interpretation [9] engine by defining appropriate abstractions and domains of abstractions for analysing a program. Deductive verification techniques, model checking and abstract interpretation analyse programs or systems which are already built and we call this the *a posteriori* approach where the process of analysis tries to extract semantic information from the text of the program or the system.

The correct-by-construction approach [10] advocates the development of a program using a process which is proof-guided or proof-checked and which leads to a correct program. This is an *a-priori* verification approach. These proof-based development methods [11] integrate formal proof techniques in the development of software and/or systems. The main idea is to start with a very abstract model of the system under development. Details are gradually added to this first model by building a sequence of more concrete ones. The relationship between two successive models in this sequence is that of refinement [11]

The essence of the refinement relationship is that it preserves already proven system properties including safety properties and termination. At the most abstract level it is obligatory to describe the static properties of a model's data by means of an invariant predicate. This gives rise to proof obligations relating to the consistency of the model. These are required to ensure that data properties which are claimed to be invariant are preserved by the events or operations of the model.

The Event B Method [12] is a refinement-based, correctness-by-construction approach for the development of event-based systems (or, more generally, event-based models). Several case studies [13] show that the Event B method provides a flexible framework for developing complex systems in an incremental and proof-based style. Since refinement necessitates checking proof obligations, an idea is to introduce concepts of reusability and instantiation of Event B models [14,15]: making it possible to re-apply already developed and proven models.

Refinement is a critical step in formal design: as we move from the abstract to the concrete we transform our requirements into an operational solution. Without refinement, the correctness of non-trivial design steps is usually a computational intensive (often intractable) problem. Refinement allows us to split the design phase into a sequence of refinement steps, each of which is proven correct through the discharging of proof obligations. When the sequence is well-engineered, this can often be done in an automated fashion. The role of the software engineer is to "find" such a sequence. The purpose of this research is to aid the engineer in this task.

4.2 Explicit Semantics of Modelling Domains and Domain Ontologies

According to Gruber [4], an ontology is a specification of a conceptualisation. An ontology can be considered as the modelling of domain knowledge. Nowadays, ontologies are used in many diverse research fields and several proposals for ontology models and languages and corresponding operational systems have been developed in the last decade. The main characteristics of an ontology are: being formal and consensual and offering referencing capabilities.

As both an ontology and a conceptual model define a conceptualization of a part of the world, we must clarify their similarities and differences. Conceptual models respect the formal criterion. Indeed, a conceptual model is based on a rigorously formalized logical theory and reasoning is provided by view mechanisms. However, a conceptual model is application requirement driven: it **prescribes** and **imposes** which information will be represented in a particular application (logical model). Thus, conceptual models do not fulfil the consensual criterion. Moreover, an identifier of a conceptual model defined concept is a name that can be referenced only inside the context of an Information System. Thus, conceptual models also do not fulfil the capability to be referenced criterion [16].

Several ontology models — like OWL [17] and KAON [18] for description logic, and PLIB [19] and MADS [20] for database design — are based on constructors provided by conceptual models based on either database or knowledge base models. These models add other constructors that facilitate satisfaction of the consensual criterion (context definition, multi-instantiation) and the capability to be referenced criterion.

Within our approach, it is clear that re-usable domain knowledge can, and should, be integrated into the modelling of a system's environment. A problem with current modelling approaches is that this knowledge is often distributed between the inside and the outside of a system in an ad-hoc fashion. By formalising the notion of ontology we can encourage (perhaps oblige) system engineers to be more methodological in how they structure and re-use their ontologies. We are currently investigating whether this can be done through a better integration of existing ontology models into our Event-B framework (through annotations) or whether we need to build a re-usable ontological framework in Event-B, motivated by the aspects of existing ontology models that we have found useful in our case studies.

4.3 Properties and Methodology

The separation of implicit from explicit gives rise to many difficulties with respect to the methodological aspects of developing software and/or systems, but it opens up many opportunities with respect to the types of properties that can be handled more elegantly. Building on the notion of functional correctness — where a software and/or a system must be verified to meet its functional requirements when executing in a well-behaved environment — we must consider the issue of system reliability being compromised. In such circumstances we would like the behaviour to degrade in a controllable, continuous, manner rather than having a non-controllable abrupt crash. One of the advantages of the proposed approach is that we can automatically distinguish between problems due to an environment which is not behaving as expected (where the system makes an assumption about its environment which is false some time during execution) and an internal fault (where the environment makes some assumption about the system which is false some time during execution). We can also automatically execute some self-healing mechanism that is guaranteed — through formal verification — to return the system and its environment to a safe, stable state.

The key to our methodology is the integration of the implicit and explicit modelling, which is shown in figure 3. The architecture should be generally applicable to the development of software and/or systems in a wide range of problem domains. This needs to be validated through application of the tools and techniques in a range of case studies that go beyond the case study presented in this paper.

The development approach may also be considered generic with respect to its application using a variety of formal techniques rather than specific to a single (set of) method(s).

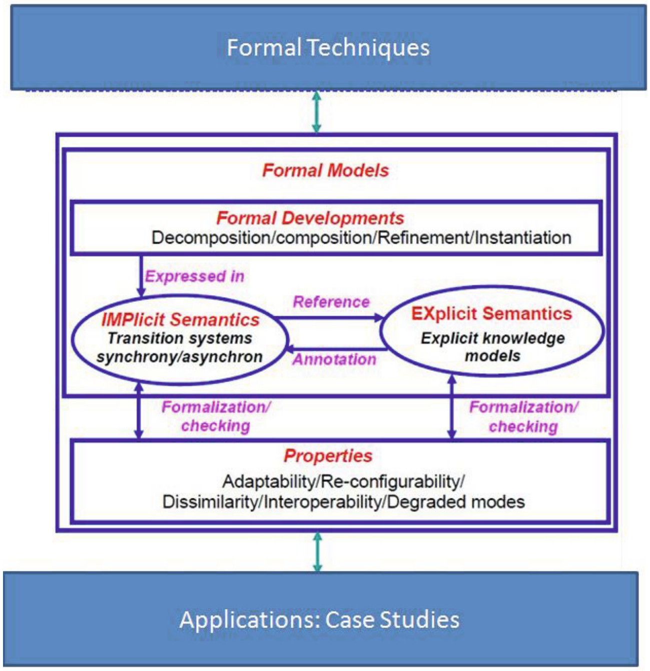


Fig. 3. A research architecture

As a minimum, the formal models must offer mechanisms for (de)composing systems, as well as for refinement and instantiation. The separation of implicit and explicit semantics is critical to independent development and verification. Furthermore, as illustrated in the case study of this paper, their integration must be directly supported by the development architecture: the implicit models will reference the explicit semantics which will provide annotations for the operational models. The best means of supporting this integration require further work: especially when we consider that combining top-down and bottom-up approaches in a formal development process is already very challenging. Initial work has shown us that the combination of different software development techniques (in particular, refinement with composition) is a major challenge. This is normally done in an ad-hoc manner, where the many different composition mechanisms lead to sate explosion problems when analysing behaviour. We propose working in an algebraic manner, which will constrain the ways in which composition can be performed: making our methodology simpler and more amenable to automated verification. A simplification of the semantics of composition risks reducing the expressiveness of our language, but we argue that finding the balance is a key part of making our approach both sound, in theory, and applicable, in practice. This balance is necessarily different when considering implicit and explicit aspects of modelling. Without a separation of these concerns, we risk a compromise which helps in the development of neither. Through separation, we can better balance the modelling of each.

5 Conclusions

We have argued that many problems in the development of correct software and/or systems could be better addressed through the separation of implicit and explicit semantics. The key idea is to re-formalize correctness as a ternary (rather than binary) relation.

We have proposed that traditional formal methods need to be better integrated with ontology models, in order to support a clearer separation of concerns.

Through a simple example, we have illustrated how ontological semantics can be specified using Event B contexts and that this information can be integrated with the behavioral requirements — in an incremental fashion — through refinement. The simple example addresses the simple problem of information interchange. (A good example of the consequences of not modelling this formally can be found in the report on the Mars Climate Orbiter [21] where confusing imperial and metric measurements caused a critical failure.)

A main contribution of the paper is to place the implicit-explicit structure within the context of the relevant state-of-the-art. We emphasise the importance of building on well-established formal methods, treating domain/context ontology models as first-class citizens during development, and the need for a pragmatic approach that integrates into existing methods in a unified and coherent fashion.

Finally, we give an overview of where we think further research needs to be done, formulating the goals as the need for an architecture of inter-related research tasks.

References

1. Levesque, H.J.: A logic of implicit and explicit belief. In: Brachman, R.J. (ed.) AAAI, pp. 198–202. AAAI Press (1984)
2. Uschold, M.: Where are the semantics in the semantic web? AI Mag. 24, 25–36 (2003)
3. van Lamsweerde, A., Willemet, L.: Inferring declarative requirements specifications from operational scenarios. IEEE Trans. Softw. Eng. 24, 1089–1114 (1998)
4. Gruber, T.R.: A translation approach to portable ontology specifications. Knowl. Acquis. 5(2), 199–220 (1993)
5. Ait-Ameur, Y., Méry, D.: Handling heterogeneity in formal developments of hardware and software systems. In: Margaria, T., Steffen, B. (eds.) ISoLA 2012, Part II. LNCS, vol. 7610, pp. 327–328. Springer, Heidelberg (2012)

6. Garlan, D., Schmerl, B.: Model-based adaptation for self-healing systems. In: Proceedings of the First Workshop on Self-healing Systems, WOSS 2002, pp. 27–32. ACM, New York (2002)
7. Bjorner, D.: Software Engineering 1 Abstraction and Modelling; Software Engineering 2 Specification of Systems and Languages, Software Engineering 3 Domains, Requirements, and Software Design. Texts in Theoretical Computer Science. An EATCS Series. Springer (2006)
8. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press (2000)
9. Cousot, P., Cousot, R.: Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: POPL, pp. 238–252 (1977)
10. Leavens, G.T., Abrial, J.R., Batory, D., Butler, M., Coglio, A., Fisler, K., Hehner, E., Jones, C., Miller, D., Peyton-Jones, S., Sitaraman, M., Smith, D.R., Stump, A.: Roadmap for enhanced languages and methods to aid verification. In: Proceedings of the 5th International Conference on Generative Programming and Component Engineering, GPCE 2006, pp. 221–236. ACM, New York (2006)
11. Back, R.J.R.: On correct refinement of programs. Journal of Computer and Systems Sciences 23(1), 49–68 (1981)
12. Abrial, J.R.: Modeling in Event-B - System and Software Engineering. Cambridge University Press (2010)
13. Abrial, J.R., Cansell, D., Méry, D.: A mechanically proved and incremental development of ieee 1394 tree identify protocol. Formal Asp. Comput. 14(3), 215–227 (2003)
14. Abrial, J.R., Hallerstede, S.: Refinement, decomposition, and instantiation of discrete models: Application to event-b. Fundam. Inf. 77(1-2), 1–28 (2007)
15. Cansell, D., Gibson, J.P., Méry, D.: Refinement: A constructive approach to formal software design for a secure e-voting interface. Electr. Notes Theor. Comput. Sci. 183, 39–55 (2007)
16. Jean, S., Pierra, G., Aït-Ameur, Y.: Domain ontologies: A database-oriented analysis. In: Cordeiro, J.A.M., Pedrosa, V., Encarnação, B., Filipe, J. (eds.) WEBIST (1), pp. 341–351. INSTICC Press (2006)
17. Bechhofer, S., Van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D., Patel-Schneider, P., Stein, L., et al.: Owl web ontology language reference. W3C recommendation 10, 2006-01 (2004)
18. Bozsak, E., Ehrig, M., Handschuh, S., Hotho, A., Maedche, A., Motik, B., Oberle, D., Schmitz, C., Staab, S., Stojanovic, L., et al.: Kaon—towards a large scale semantic web. E-Commerce and Web Technologies, 231–248 (2002)
19. Pierra, G.: Context-explication in conceptual ontologies: the plib approach. In: Proceedings of the 10th ISPE International Conference on Concurrent Engineering (CE 2003). Enhanced Interoperable Systems, vol. 26, p. 2003 (2003)
20. Parent, C., Spaccapietra, S., Zimányi, E.: Spatio-temporal conceptual models: data structures + space + time. In: Proceedings of the 7th ACM International Symposium on Advances in Geographic Information Systems, GIS 1999, pp. 26–33. ACM, New York (1999)
21. Stephenson, A., Mulville, D., Bauer, F., Dukeman, G., Norvig, P., LaPiana, L., Rutledge, P., Folta, D., Sackheim, R.: Mars climate orbiter mishap investigation board phase I report. Technical report, NASA, Washington, DC (1999)

The Technological and Interdisciplinary Evolution in Machine and Plant Engineering – Industry 4.0

Axel Hessenkämper

GEA Westfalia Separator Group GmbH

In machine and plant engineering, the terms Industry 4.0, Internet of Things (IoT) and Cyber Physical Systems (CPS) are currently very present and are discussed, defined and described differently.

To meet future requirements of machines and plants it is necessary to combine modern and state of the art technologies of information technology with the classic, industrial processes and systems. Today, 80% of the innovations in production technology go back to the integration of information technology in mechanical engineering. [1]

The reduced complexity in machine and process operation (usability, simplicity and flexibility), a fast and continuous analysis of data and a simple diagnosis of the machinery and equipment are only examples of a future production. Not only the machines and processes itself have to be user friendly and more flexible, the engineering of the systems have to face the same challenge. Engineers need manageable and complexity reduced high capable tools and systems to be able to meet future requirements. Simplicity in the exchange and reusability of every software component at any time is truly component-based engineering. [3] The above mentioned points require sustainable research efforts to create systems that meet the actual demands and lead to systems which in their flexibility and in the support of reducing complexity contribute manageability – and not, even again are a source of additional complexity. [2]

To apply the techniques of information technology to the manufacturing and development of the classic processes and systems requires an interdisciplinary exchange in order to create a mutual understanding. Information technology needs to have a rough understanding of the requirements of machine design technology and vice versa. Interdisciplinary also means that workers with their skills and experience will be more involved in both, the creative design and planning process as well as the operational environment. [2]

As part of the Industrial Day 2014, first interdisciplinary applications and ideas of information technology combinations with classical processes and systems, as well as further requirements are presented. Three major world leading automation equipment vendors are presenting their Industry 4.0 approaches and how they interpret the evolutionary steps towards it.

The way towards the Production of tomorrow by the use of scientific automation systems and platforms will be presented by Mr. Gerd Hoppe from Beckhoff Automation GmbH.

Dr. Detlef Pauly, Siemens AG, will show some brief explanations concerning the background of Industry 4.0 and present the trend in manufacturing and automation towards a Digital Enterprise and how this is addressed by companies and major automation vendors.

An increase of capabilities in industrial control systems is raising the complexity of the systems. The acceptance by the users of these complex systems is only guaranteed, if the simplicity is increasing too. Diego Escudero from Omron Electronics will present a robotics function in combination with a vision system as an example of simple programming and high flexibility automation system.

References

1. Bundesministerium für Bildung und Forschung: Zukunftsbild “Industrie 4.0”. Bonn (2014)
2. Kagermann, W.H.: Umsetzungsempfehlung für das Zukunftsprojekt Industrie 4.0. Büro der Forschungsunion im Stifterverband für die Deutsche Wissenschaft e.V., Berlin (2012)
3. Margaria, T., Steffen, B.: Simplicity as a Driver for Agile Innovation. IEEE Computer 43(6), 90–92 (2010)

Integrated Code Motion and Register Allocation

Gergö Barany*

Vienna University of Technology

1 Problem and Research Question

Code motion and register spilling are important but fundamentally conflicting program transformations in optimizing compiler backends. Global code motion aims to place instructions in less frequently executed basic blocks, while instruction scheduling within blocks or regions (all subsumed under 'code motion') arranges instructions such that independent computations can be performed in parallel. These optimizations tend to increase the distances between the definitions and uses of values, leading to more overlaps of live ranges. In general, more overlaps lead to higher register pressure and insertion of more expensive register spill and reload instructions in the program. Eager code motion performed before register allocation can thus lead to an overall performance decrease.

On the other hand, register allocation before code motion will assign unrelated values to the same physical register, introducing false dependences between computations. These dependences block opportunities for code motion that may have been legal before assignment of registers. This is an instance of the *phase ordering problem*: Neither ordering of these phases of code generation provides optimal results. A common way to sidestep this problem is by solving both problems at once in an integrated fashion.

The aim of this thesis is to develop a fully integrated approach to global code motion and register allocation. The expected result is an algorithm that determines an arrangement of instructions that leads to minimal spill code while performing as much global and local code motion as possible.

The expected contributions of the method developed in this thesis are twofold: First, the only *global* algorithm that integrates code motion/scheduling issues and register allocation on the scope of whole functions; second, an algorithm that is guided by the *exact* needs of the register allocator rather than by an estimate of register pressure computed beforehand.

The algorithm can be tuned to give higher preference to either code motion or spilling; by evaluating various settings of the tuning parameter, we can find the best trade-off between these two conflicting phases for any given input program. It is expected that preferring minimal spilling is often, but not always, the best choice.

* This work was supported by the Austrian Science Fund (FWF) under contract P21842, *Optimal Code Generation for Explicitly Parallel Processors*, `http://www.complang.tuwien.ac.at/epicopt/`.

2 Related Work

The proposed technique has some similarities to previous integrated scheduling and register allocation approaches by Norris and Pollock [3], Berson et al. [2], and Touati [4]. All of these add ordering arcs to data dependence graphs to ensure register reuse. However, all of these techniques schedule only within basic blocks. As such, they are also limited to using estimated register pressure information at the start and end of each block; in contrast, the proposed global technique developed for this thesis will tightly integrate the register allocator's exact model of the spilling problem with its scheduling choices.

3 Methods

The effectiveness of the approach will be demonstrated by an implementation in the LLVM compiler framework, evaluated using the standard SPEC CPU benchmark suite. The relevant quantity to be evaluated is the change in program execution time.

4 Preliminary Results

Careful benchmarking of code generated by the algorithm showed that some speedups are possible this way, although on average, the effect is small [1]. Some benchmarks are slowed down because minimal spilling is not the optimal choice if it blocks code motion opportunities.

5 Next Steps

As the preliminary results showed, optimizing code motion for minimal spilling is not always the best trade-off. Tolerating some spills in exchange for more aggressive code motion opportunities sometimes results in much faster code overall. A parameter capturing this trade-off was added to the problem formulation, and the search for the best parameter value for each benchmark is currently in progress.

References

1. Barany, G., Krall, A.: Optimal and heuristic global code motion for minimal spilling. In: Jhala, R., De Bosschere, K. (eds.) Compiler Construction. LNCS, vol. 7791, pp. 21–40. Springer, Heidelberg (2013)
2. Berson, D.A., Gupta, R., Soffa, M.L.: Integrated instruction scheduling and register allocation techniques. In: Carter, L., Ferrante, J., Sehr, D., Chatterjee, S., Prins, J.F., Li, Z., Yew, P.-C. (eds.) LCPC 1998. LNCS, vol. 1656, pp. 247–262. Springer, Heidelberg (1999)
3. Norris, C., Pollock, L.L.: A scheduler-sensitive global register allocator. In: Supercomputing 1993, pp. 804–813 (1993)
4. Touati, S.A.A.: Register saturation in superscalar and VLIW codes. In: Wilhelm, R. (ed.) CC 2001. LNCS, vol. 2027, pp. 213–228. Springer, Heidelberg (2001)

On the Algebraic Specification and Verification of Parallel Systems[⋆]

Nikolaos Triantafyllou, Katerina Ksystra, and Petros Stefaneas

National Technical University of Athens

1 Problem and Research Question

Observational Transition Systems (OTS) are a rich subclass of behavioral algebraic specifications, used to model, specify and verify distributed systems. Usually, due to the abstraction level of these specification techniques in order to obtain an implementation, refinement of the original specification is required. However, existing refinement methodology was not designed to support concurrent systems and indeed its use can lead to un-intentional semantics.

2 Related Work

Most of the existing related work is concerned with reasoning about programs where two or more processes share a memory location (for reading and writing, this is called a data race). In [1] the authors argue that because the hardware and software implementations of multi-core systems do not provide a sequentially consistent shared memory (they use relaxed memory models) the existing approaches for reasoning about concurrent programs are not applicable. To this end they provide a semantic foundation for reasoning about such programs and demonstrate the usefulness of their approach on a variety of low-level concurrency algorithms. However, their methodology does not provide support for automation and as they state can be tedious in its current form. Also, in [2] the authors develop an integrated approach to the verification of behaviorally rich programs, with proof support from the HOL4 theorem prover and deal with the issues of data-race they encounter in their case study using locks.

3 Methods

The problem of the existing refinement methodology arises because as it is clearly stated in [3] *the issue of concurrency is not addressed by their methodology, but the*

⋆ This research has been co-financed by the European Union (European Social Fund - ESF) and Greek national funds through the Operational Program ′Education and Lifelong Learning′ of the National Strategic Reference Framework (NSRF) - Research Funding Program: THALES: Reinforcement of the interdisciplinary and/or inter-institutional research and innovation

existing refinement methodology only considers serial evaluation by term rewriting. Thus, it was not meant to be used for the refinement of concurrent systems (defined using [4]). However, in many cases it is required to refine the abstract specification to reach an implementation. To counter the above stated problem we propose a novel extension of the OTS methodology called Threaded-OTS (ThOTSs) which permits the safe use of the existing refinement methodology presented in [3]. The key idea behind ThOTSs is that the state changing operations (transitions) of a behavioral object have a *start* and an *end*. Inbetween these, there exists no guarantee for the values characterizing the state (observers). We can only safely reason that the value of the observers will either have changed or it will have not. To capture this, the observers must stop being deterministic once a transition has started executing. To define this, we replace the classical OTS observers with *oracles*, boolean observation operations that return true for both the expected value of the observer after the execution of the transition and for the previous value of the observer. Only when a transition has finished executing does the system re-enter a deterministic state.

4 Preliminary Results

We demonstrated using a simple mutual exclusion system why the existing methodology is indeed not suited for the refinement of OTS specifications of concurrent distributed systems. Next, we demonstrated using the same example, that ThOTSs are safe to use with the existing refinement methodology. Finally, we presented a proof as to why ThOTSs can be safely refined using [3]. To the best of our knowledge the proposed methodology is the first OTS/CafeOBJ framework that can be used in conjunction with the refinement method of [3] to safely reason about concurrent distributed systems.

5 Next Steps

In the future we intend to better evaluate the applicability of the proposed framework on real life systems by conducting case studies. Additionally, we wish to develop alternative refinement methods that will be safe to use for all behavioral specifications (including concurrent ones).

References

1. Owens, S.: Reasoning about the implementation of concurrency abstractions on x86-TSO. In: D'Hondt, T. (ed.) ECOOP 2010. LNCS, vol. 6183, pp. 478–503. Springer, Heidelberg (2010)
2. Ridge, T.: Verifying distributed systems, the operational approach. In: Proceedings of the 36th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pp. 429–440 (2009)
3. Malcolm, G., Goguen, J.A.: Proving Correctness Of Refinement And Implementation, Technical Monograph, Oxford University Computing Laboratory (1994)
4. Diaconescu, R.: Behavioural specification for hierarchical object composition. Theoretical Computer Science 343(3), 305–331 (2005)

Property-Specific Benchmark Generation

Maren Geske

TU Dortmund

1 Problem and Research Question

There are countless software verification tools most of which specialize in a very narrow set of tasks to master a certain class of verification problems. As a consequence, many of the benchmark prolems these tools are typically evaluated on focus on a small set of phenomena. This way, tool developers miss the opportunity to assess, optimize and test their tools in a broader context. Moreover, it makes it extremely difficult for prospective users to identify the tool most suitable for the problem setting they are facing.

The aim of my research is to develop a framework that generates benchmarks of a given complexity profile from a set of properties of tailored complexity. The benchmarks can be used to compare tools concerning their abilities, but can also be used to improve or optimize the verification process. The generation process is adjustable in form of restrictions concerning program complexity and supported language constructs as well as the programming language in general. It is designed to also emit a comprehensive “solution” in form of satisfied and unsatisfied LTL formulae and reachability specifications that describe the program behavior. The benchmarks are generated from automata synthesized from a given specification that characterizes the behavioral properties of the program. The program logic is crafted by encoding the automaton states using variables and data structures such that transitions are realized by variable assignments and data restructuring. The resulting code structure of an automaton is then further altered by property-preserving transformations, ensuring that the program remains in conformance to the specification.

As there are countless possibilities to design generation constraints, the variety of supported transformations is constantly increasing, but will only ever remain a fraction of all possible transformations. Currently the framework supports arithmetic expressions, value ranges and simple array operations. Internally, code motion [3] is used to break the correlation between code blocks and automaton transitions. A constant dialogue with the verification community is desired to make the generator more useful for all engineers and users.

2 Related Work

There are several benchmark suites generators being built for specific problem fields. However, to the best of my knowledge there is no approach which aims at generating a benchmark suite for a wide range of different software verification tools. Related research are nontheless obfuscation techniques in general [1] and methods like code motion [3] to alter the control flow of the programs.

3 Methods

The benchmarking suite is regularly used to create test problems for the RERS program validation and verification challenge [2]. New ideas and suggestions are discussed with the participants before being integrated into the generator. This ensures that the benchmarks built with this framework realize desired features of a benchmark generator. To ensure the correctness of the generated programs concerning their specification, a test suite is being built internally that is backed by active automata learning [5].

4 Preliminary Results

Generated programs in C and Java were used in three editions of RERS from 2012 on and were therefore pedantically analyzed by several verification tools. The participants acknowledged the problems as “challenging”, since they showed some weaknesses in their tools and helped to improve the verification process.

5 Next Steps

As the first experiments at RERS showed it is fairly easy to create big and complex programs. However, the goal is to create structurally interesting problems that are similar to real code, meaning they should include a lot of different structures that are common in handwritten code. The next milestone is therefore to include restructuring functions for arrays as simple objects and complex code structures in form of custom data structures. The whole framework should finally be highly parametrized concerning used code elements [4] and support numerous frequently used programming languages.

References

1. Collberg, C., Thomborson, C., Low, D.: A taxonomy of obfuscating transformations. Technical Report 148, Department of Computer Science, University of Auckland (July 1997)
2. Howar, F., Isberner, M., Merten, M., Steffen, B., Beyer, D.: The RERS grey-box challenge 2012: Analysis of event-condition-action systems. In: Margaria, T., Steffen, B. (eds.) ISoLA 2012, Part I. LNCS, vol. 7609, pp. 608–614. Springer, Heidelberg (2012)
3. Knoop, J., Rüthing, O., Steffen, B.: Lazy code motion. In: Proceedings of the ACM SIGPLAN 1992 Conference on Programming Language Design and Implementation, PLDI 1992, pp. 224–234. ACM, New York (1992)
4. Neubauer, J., Steffen, B., Margaria, T.: Higher-order process modeling: Product-lining, variability modeling and beyond. In: Festschrift for Dave Schmidt, pp. 259–283 (2013)
5. Steffen, B., Howar, F., Merten, M.: Introduction to active automata learning from a practical perspective. In: Bernardo, M., Issarny, V. (eds.) SFM 2011. LNCS, vol. 6659, pp. 256–296. Springer, Heidelberg (2011)

Steering Active Automata Learning with Model-Driven Development

Oliver Bauer

TU Dortmund, Chair for Programming Systems
Otto-Hahn-Str. 14
D-44227 Dortmund, Germany

1 Problem and Research Question

The last decades showed that *Active Automata Learning* (AAL) can be applied successfully to a broad range of real-life systems to overcome the situation of missing adequate models that are necessary for many formal verification techniques or model-based testing approaches to automatically generate test suites.

Through *Model-Driven Development* (MDD) it is possible that domain experts are able to express workflows in a model-based fashion for some resp. domain without programming skills. In an extreme occurrence (cf. the XMDD approach [2]) it include benefits like on-the-fly model verification to minimize architectural failures and supports full code generation to create executable application drafts.

My research will focus on improving the interplay of AAL and XMDD to enhance the applicability of AAL for domain experts that are not meant to be programmers. The overall approach is intended to lead the domain expert from the beginning through different steps like generation of learning setups, usage of domain-specific filter techniques for inference acceleration, code generation and support for manual test sequence evaluation during the process execution.

2 Related Work

The libalf library [1] is an open source automata learning framework containing various active and passive learning algorithms for inferring finite-state automata. For real-life applications a comprehensive set of filtering techniques is necessary to infer models of evolving software products [6]. With our new open source *LearnLib* [1] those already have been applied successfully with ease [7].

The *Eclipse Modeling Framework* (EMF) rapidly receives growing interest for developing software in a model driven way but the corresponding diversity of APIs elongate the resp. training period to get productive. Our new *jABC4* [5] nicely integrates with modern technologies and allows developing applications in an XMDD fashion without the hurdles of long training periods to get productive.

LearnLib Studio [4] is based on previous versions of LearnLib and jABC and allows some of my research intend up to some extent. Especially in leading the domain expert through the whole process i'd like to improve previous work.

[1] `http://www.learnlib.de`

3 Methods

My work will focus on improving the usability of AAL and MDD for domain experts in a comprehensible way. The result will be evaluated along different case studies on a modern technology stack to show the ease of the taken approach.

4 Preliminary Results

Fig. 1 depicts a generic learning loop that has already been applied in simulated environments and real-life applications. The overall learning pattern (search and refinement phase) is itself purely generic such that is works in many environments and different configurations. Depending on the abstraction a user wants to see the whole process could also be used as a "learning loop" building-block.

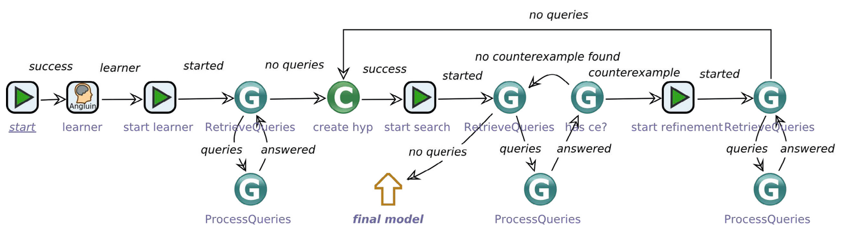


Fig. 1. Generic learning loop of LearnLib created with jABC4

5 Next Steps

Providing model based domain-specific optimizations, supporting full code generation and a guidance through the whole process along different case studies of the approach will conclude the proposed research topic. The developed strategy will illustrate the ease of bringing together AAL and XMDD while still illustrating the gain and flexibility of the involved technologies [3]. The proposed research intends to establish a basis for modern testing technology in a model-driven and automatic way fleshed out by concrete real life case studies that could be made by domain experts that are possibly non-programmers.

References

1. Bollig, B., Katoen, J.-P., Kern, C., Leucker, M., Neider, D., Piegdon, D.R.: libalf: The Automata Learning Framework. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 360–364. Springer, Heidelberg (2010)
2. Margaria, T., Steffen, B.: Agile IT: Thinking in User-Centric Models. In: Margaria, T., Steffen, B. (eds.) ISoLA 2008. CCIS, vol. 17, pp. 490–502. Springer, Heidelberg (2008)
3. Margaria, T., Steffen, B.: Simplicity as a Driver for Agile Innovation (2010)

4. Merten, M., Steffen, B., Howar, F., Margaria, T.: Next Generation Learnlib. In: Abdulla, P.A., Leino, K.R.M. (eds.) TACAS 2011. LNCS, vol. 6605, pp. 220–223. Springer, Heidelberg (2011)
5. Neubauer, J.: Higher-Order Process Engineering. Phd thesis, TU Dortmund (2014)
6. Steffen, B., Howar, F., Merten, M.: Introduction to Active Automata Learning from a Practical Perspective. In: Bernardo, M., Issarny, V. (eds.) SFM 2011. LNCS, vol. 6659, pp. 256–296. Springer, Heidelberg (2011)
7. Windmüller, S., Neubauer, J., Steffen, B., Howar, F., Bauer, O.: Active continuous quality control. In: CBSE, pp. 111–120 (2013)

Generation of Domain-Specific Graphical Development Tools Targeting Heterogeneous Platforms

Michael Lybecait and Dawid Kopetzki

TU Dortmund University
Department of Computer Science
Chair for Programming Systems

1 Problem and Research Question

Nowadays, model-driven engineering (MDE) paradigms are used in software development since the systems under development become larger and more and more complex. The main concept of these paradigms is the specification of an abstraction of the system called model. In a later development step the model is used to generate code which eases the development of the system. Popular model-driven engineering paradigms are *Model-Driven Architecture* (MDA) [6] developed by the *Object Management Group* [12] and the *Eclipse Modeling Framework* [1] which consist of several modeling frameworks and tooling. However, a significant drawback of these model-driven paradigms is the lack of efficient synchronization mechanisms between the models and code. Furthermore, MDA which is used for model-driven development of large heterogeneous systems suffer from further problems [3]:

– MDA requires the definition of several layers of models through which all system changes have to be propagated which is error-prone and costly.
– To synchronize changes between model and code some kind of round trip engineering is required.
– MDA approaches enforces either one monolithic tool which is hard to adopt for new demands or several tools arranged in a chain which have to be smoothly integrated in the development process.

Furthermore, this general development approaches require the developer's knowledge of both the application domain as well as the target platform. Additionally, the generality of MDA makes full code generation of tools impossible.

The ultimate goal would be to develop a metamodeling framework that facilitates fully automatic generation of fully functional modeling tools for different platforms. As input for this framework should serve an abstract tool specification provided by a developer who only has knowledge about the tool's domain.

The first step towards such a tool is the restriction to a specific domain. As a consequence, a domain-specific language (DSL) can be provided for the definition of an abstract tool specification. Unfortunately, a complete specification of a modeling tool, especially the functional behavior, would blow-up the domain-specific language. Therefore, we want to follow the simplicity approach [9] by generating the majority of code and additionally, providing an interface to extend the tool's functionality on a domain-specific level. Furthermore, an abstract specification of target platforms would provide the deployment of the tool for a wide filed of application without the need of an application expert.

2 Related Work

Several approaches exist which aim to solve the problems mentioned in the previous section. The authors in [2] propose an source code generator for large multilayer architectures. However, this approach is meant to support a software engineer by generating specific code snippets which have to be manually integrated into the software. Kelly and Tovanen present in [5] the domain-specific modeling (DSM) approach which enables full code generation. Typically, these applications are generated for one specific platform. As the DSM, our approach should allow for the specification of full code generable applications with the possibility of functional extendability on a domain-specific level. able applications with the possibility of functional extendability on a domain-specific level. In addition, we plan to add an abstract, platform independent specification of target platforms for which the application should be generated.

3 Methods

With the Cinco *metamodeling suite* [10] one can define an abstract tool specification for modeling tools based on graph-like models. The tool generation is implemented by means of the *Java Application Building Center* [8][11]. Cinco is realized within the *Eclipse Modeling Framework* (EMF) which provides several advantages such as a metamodeling language (*Ecore*) with ready-to-use persistence mechanisms and frameworks which ease the development of different editors (textual/graphical) in the Eclipse context. By building up on Ecore, TransEM [7] can be used to generate domain-specific SIBs for each kind of domain corresponding to Cinco's metamodel. Consequently, the development in jABC and the usage of a meta-metamodel for all abstract tool specifications created by Cinco ensures an agile software development process of the Cinco framework. Cinco is itself a domain specific tool for creating domain specific tools. By applying the methods on the Cinco development process, we continually improve Cinco. This will help the development of domain specific tools in the future.

4 Preliminary Results

As a proof of concept, first modeling tools were developed with Cinco, i.a. tools for modeling timed automata, probabilistic timed automata, Markov decision processes , labeled transition systems and petri nets. Some of these tools include a code generator. Using the TransEM in combination with code generator frameworks such as Genesys [4] the code generator itself can be modeled with the *jABC*.

5 Next Steps

As mentioned earlier the Model-Driven Architecture [6] describes several types of models, which are the platform independent model (PIM), the platform specific model (PSM) and the code. A tool that is to be run on various platforms, can be described in a PIM. This PIM has then to be transformed to several PSMs that confirm to the wished platform architectures (PA). If there is a metamodel such platform architectures and a metamodel for the PIM in question, TransEM can be used to generate a transformation language, that can be used to describe a transformation from a PIM to a PSM for a given platform architecture. Figure 1 shows the generation of a PSM from a PIM given a platform architecture specification. With Cinco exists a domain specific language (DSL) to define a PIM for the tool. Further work has to be done to define a DSL for platform architecture specifications.

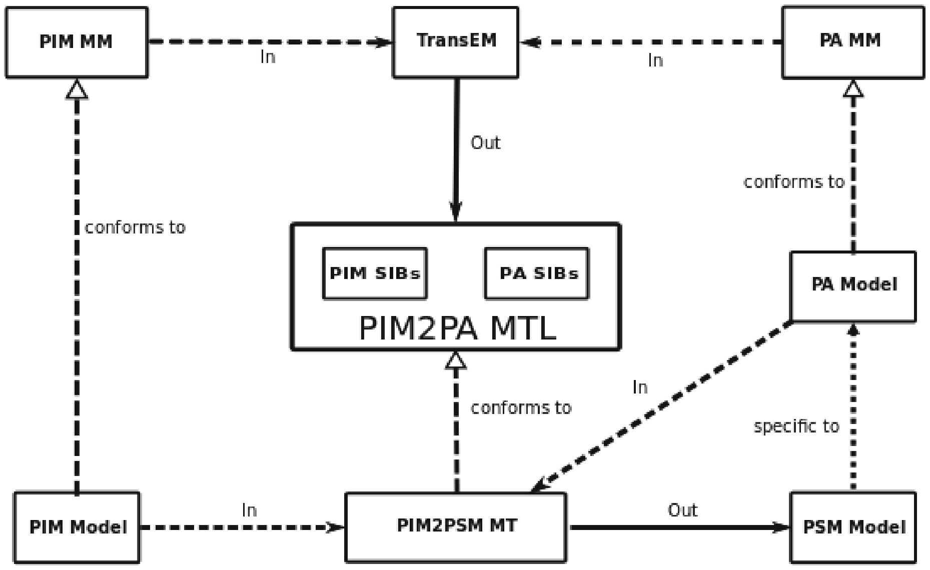


Fig. 1. MTL for PSM from PIM and Platform architectures

References

1. Eclipse Modeing Framework, http://www.eclipse.org/modeling/emf/
2. Altiparmak, H.C., Tokgoz, B., Balcicek, O.E., Ozkaya, A., Arslan, A.: Source code generation for large scale applications. In: 2013 International Conference on Technological Advances in Electrical, Electronics and Computer Engineering (TAEECE), pp. 404–410. IEEE (2013)
3. Herrmann, C., Krahn, H., Rumpe, B., Schindler, M., Völkel, S.: Scaling-up model-based-development for large heterogeneous systems with compositional modeling. In: Arabnia, H.R., Reza, H. (eds.) Software Engineering Research and Practice, pp. 172–176. CSREA Press (2009), http://dblp.uni-trier.de/db/conf/serp/serp2009.html#HerrmannKRSV09
4. Jörges, S. (ed.): Construction and Evolution of Code Generators. LNCS, vol. 7747. Springer, Heidelberg (2013)
5. Kelly, S., Tolvanen, J.P.: Domain-Specific Modeling: Enabling Full Code Generation. Wiley-IEEE Computer Society Press, Hoboken (2008)
6. Kleppe, A.G., Warmer, J., Bast, W.: MDA Explained: The Model Driven Architecture: Practice and Promise. Addison-Wesley Longman Publishing Co., Inc., Boston (2003)
7. Lybecait, M.: Entwicklung und Implementierung eines Frameworks zur grafischen Modellierung von Modelltransformationen auf Basis von EMF-Metamodellen und Genesys (2012)
8. Margaria, T., Steffen, B.: Business Process Modelling in the jABC: The One-Thing-Approach. In: Cardoso, J., van der Aalst, W. (eds.) Handbook of Research on Business Process Modeling. IGI Global (2009)
9. Margaria, T., Steffen, B.: Simplicity as a Driver for Agile Innovation. Computer 43(6), 90–92 (2010)
10. Naujokat, S., Lybecait, M., Steffen, B., Kopetzki, D., Margaria, T.: Full Generation of Domain-Specific Graphical Modeling Tools: A Meta^{2}modeling Approach (under submission, 2014)
11. Neubauer, J., Steffen, B., Margaria, T.: Higher-Order Process Modeling: Product-Lining, Variability Modeling and Beyond. Electronic Proceedings in Theoretical Computer Science 129, 259–283 (2013)
12. Object Management Group (OMG): Object Management Group, www.omg.org, (last accessed May 20, 2014)

Living Canvas

Barbara Steffen

University of Twente

1 Problem and Research Question

Nowadays strategic analyses are important for companies and organizations to survive in the fast developing and moving environments of today. To support managers analysis templates have been developed which give a static overview of the most important factors when thinking of new strategies and analysing the current circumstances of the organization and its market exploitation potential. However, these templates are static and only designed to structure manual textual input.

But how far can this state of the art be extended by exploiting automated means know from computer science for analysing and controlling the interaction with the canvas? Is it possible to better guide the user, prohibit misuse, and to generate valuable feedback and recommendations?

2 Related Work

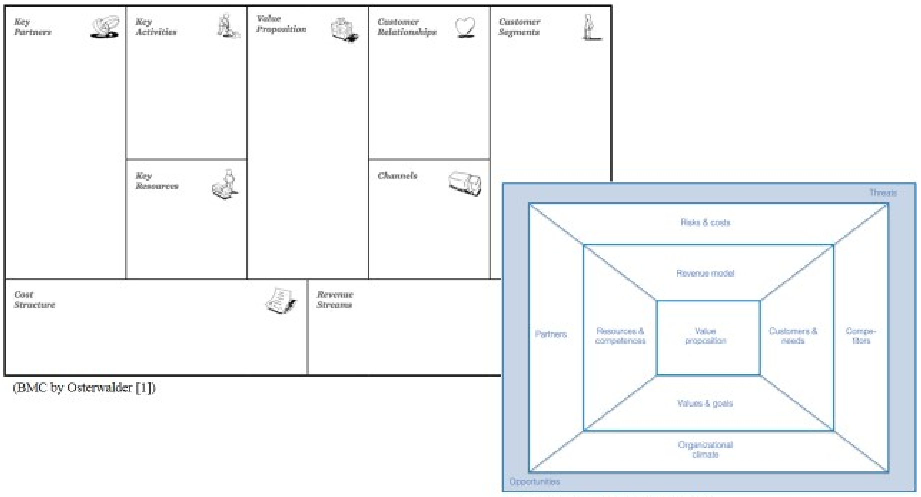
By developing and defining nine general building blocks meant to be usable in all organizations and situations the *Business Model Canvas* (BMC) by Osterwalder [1] was a break through. It is a widely used template supporting the development of business strategies. The nine separate but connected building blocks form an overview of the organization's situation. The building blocks can be structured in three categories infrastructure management, product innovation and customer relationship. The purpose of the template is to remind managers of the essential factors when thinking of a valuable strategy. Thus it can be regarded as a guideline for a process of strategic innovation. Osterwalder's template is available in form of a huge sized poster or as an App providing the user with a static layout to be filled out manually.

The *Strategy Sketch* by Kraaijenbrink [2] elaborates the BMC in three respects:

It adds two 'missing' elements, the *strategic purpose* and the *competition*, and it keeps all elements and therefore the argumentation on the *same level of abstraction*. The strategy sketch consists of ten elements. Most of them match the building blocks of the BMC, but there are also some adjustments as exchanging or renaming elements.

3 Methods and Preliminary Results

Recently, the *Business Model Developer* tool has been developed by Boßelmann [3]. It is a tool inspired by the BMC that is developed with the goal to enact the currently passive canvases with computer science technology. This comprises the definition and validation of relations between various building blocks, the enforcement of adequate types of entries, as well as feedback in terms of computed results or hints concerning missing information. In addition it can be adapted to different forms of templates as the BMC or the Strategy Sketch.



(BMC by Osterwalder [1])

(Strategy Sketch by Kraaijenbrink [2])

4 Next Steps

On the basis of the current state of the Business Model Developer tool a domain specific version for the project ‘Comitato Girotondo’ [4] will be developed. To reach the state of a functioning example for this project the two templates the Business Model Canvas and the Strategy Sketch will be used in the form they were published for further comparisons and analysis. What are the main differences? Are they significant? Does it make sense to implement both into this tool and shall it even be possible to create a mix or new elements individually?

The example shall then proof that a tool is able to act as a `living´ model which supports the user in making the best decisions by considering all the information available. The goal is to fill the tool regarding this specific domain with enough information to provide a user-friendly service in a simplicity-oriented fashion [5].

References

1. Osterwalder, A., Pigneur, Y.: Business model generation. John Wiley & Sons, Inc. (2010)
2. Kraaijenbrink, J., `http://kraaijenbrink.com/blog/`
3. Boßelmann, S., Margaria, T.: Domain-Specific Business Modeling with the Business Model Developer. In: Margaria, T., Steffen, B. (eds.) ISoLA 2014, Part II. LNCS, vol. 8803, pp. 545–560. Springer, Heidelberg (2014)
4. ComitatoGirotondo, GassinoTorinese (TO), Italy, `http://www.comitato-girotondo.org/`
5. Margaria, T., Steffen, B.: Simplicity as a Driver for Agile Innovation. IEEE Computer 43(6), 90–92 (2010)

Feedback-Based Recognition of Human Identities Using Color and Depth Data

Frederik Gossen

TU Dortmund University
frederik.gossen@udo.edu

1 Motivation

Usually admission control, in places like fitness studios, is done using RFID chips these days. People have to swipe their cards to enter certain areas. This technology is a very reliable and yet somewhat comfortable way to check peoples' identities. The idea of this bachelor thesis was to improve the aspect of comfort at the cost of a relatively low loss of accuracy by recognizing human identities from visual features. The focus lies on spatial normalization of human faces and its impact on holistic facial recognition methods such as PCA. Ideally precise alignment of faces should improve recognition significantly. Also the impact of the chosen camera sensor will be evaluated.

This work was inspired and supported by sysTeam GmbH Dortmund.

2 Related Work

Significant process has been made in face recognition over the last years using both 2D and 3D data [1]. Most importantly for this thesis are holistic methods which process images as a whole [1, pp. 45–48]. Normalization steps can have a significant impact on the recognition rate here since these methods are often susceptible to illumination and face orientation. Other methods extract features from the data which can be more robust to varying conditions [1, pp. 43–44]. Both approaches can be applied to 2D and 3D data.

3 Problem and Research Question

This thesis focuses on extraction and spatial normalization of human faces from color and depth sensor data. In particular frontal face images are generated from potentially rotated views of a person's face. Two different cameras and one Microsoft Kinect are used as sensors to capture individuals. The sensors' information will be merged and the face of interest will be cropped out of the data. It will then be aligned and processed with holistic facial recognition methods.

In a first step the depth images are transformed into a point cloud in euclidean space. Color information is then added either from the Kinect's camera itself or from one of the external cameras. In the second case camera calibration parameters have to be learned from annotated sample images.

In a next step faces are detected in camera images using a Haar Fature-based Cascade Classifier. This generally results in more than one face, since the classifier is tuned to have a high detection rate at the cost of a relatively high false alarm rate. Hence the main face of interest has to be selected in a further step using additional depth information. Multiple features are evaluated to rate each of the candidate faces. Finally the highest rated face is cropped out of the point cloud using a sphere around the face's nose tip.

In order to estimate the head pose, facial feature points are detected in the camera images and are then projected onto the point cloud. Using the spatial information of eye and mouth corners it is possible to estimate the head pose. The information can also be used to get reference points from other regions of the face that might have less noise in depth information such as cheeks. These reference points are likely to lead to a more accurate estimation of the head pose. Other methods of head pose estimation include plane fitting on the center face [2]. Finally the face is rotated accordingly to generate a frontal view.

Since these views are generally not positioned perfectly, a final step of alignment is applied. The frontal view images are translated and rotated in order to maximize horizontal symmetry. Another criterion is to align the face according to its elliptic shape.

The impact of these normalization steps will be evaluated on a test set of 10 subjects looking in 4 different directions. Thus recognition rates for each direction and holistic method can easily be compared.

4 Outlook

Currently, extracting facial information from the captured sensor data results in frontal images of relatively low resolution. Since the camera images provide significantly more pixels than the Kinect's depth image, color information is used only partially. One next step might be to interpolate points between the captured ones and to map the known color information onto them. This will improve the resolution of computed frontal images and hence might lead to better base for recognition.

Another next step will be to extract visual features from the generated frontal faces rather than just processing them with holistic methods. These could be geometric measures but also features extracted at certain facial feature points [2] [3]. The features can be evaluated with regard to their reliability and expressiveness. Once promising features are found, they can be combined to achieve an even stronger recognition system [4] [5].

Finally both, the recognition system and the RFID technology, can be put together in a way that the RFID technology is used as a fall back method in case a person was not reliably recognized in the first place. This setup will not only improve the aspect of the access control system's comport, but it also allows to collect training data for each individual over time.

References

1. Jafri, R., Arabnia, H.: A survey of face recognition techniques. JIPS 5, 41–68 (2009)
2. Uřičář, M., Franc, V., Hlaváč, V.: Detector of facial landmarks learned by the structured output SVM. In: VISAPP 2012, pp. 547–556 (2012)
3. Wiskott, L., Fellous, J., Krüger, N., von der Malsburg, C.: Face recognition by elastic bunch graph matching. In: Sommer, G., Daniilidis, K., Pauli, J. (eds.) CAIP 1997. LNCS, vol. 1296, pp. 355–396. Springer, Heidelberg (1997)
4. Neubauer, J., Steffen, B., Margaria, T.: Higher-order process modeling: Product-lining, variability modeling and beyond. In: Festschrift for Dave Schmidt, pp. 259–283 (2013)
5. Heusch, G., Marcel, S.: Face authentication with salient local features and static bayesian network. In: Lee, S.-W., Li, S.Z. (eds.) ICB 2007. LNCS, vol. 4642, pp. 878–887. Springer, Heidelberg (2007)

Real Time Standardization Process Management

Axel Hessenkämper

University of Potsdam

1 Problem and Research Question

Today, High-Tech companies not only have to develop the most innovative products and solutions for their customers, but also to develop methods, systems and procedures for internal process improvement and lean structures. Product and Product line standardization is the most common approach to cut costs by reducing assembly time, raising scale effects in procurement, service strategies, and optimizing time to market. However, the standardization process becomes increasing complex with the number of products and the amount of parts within these products. Moreover, the ratio of Software and IT-Technology compared to physical parts within a machine is also growing. This makes standardization an expensive enterprise with difficult to predict impact, due to the complex and often unknown interdependencies within the overall production process. Thus methods are required to manage and structure the standardization process in a more transparent way, ideally allowing time and cost estimations, as well as project-accompanying audits.

The thesis aims at developing a comprehensive modeling solution that allows all stakeholders to easily follow the production process, identify their responsibilities, and act according to the imposed standardization discipline. In particular the solution should provide each stakeholder with a *tailored view* that focusses on the relevant information for their next step(s). This tailoring is important to let people with different backgrounds and responsibilities cooperate in a consistent fashion without being disturbed by information concerning other responsibility profiles. E.g. managers are provided with cost information, state of the global process, or warnings, in case some unforeseen hurdles appear, while technicians obtain their technological tasks with their individual deadlines. In fact, essential data like the actual costs, project progress and the exceedance of deadlines will be displayed and monitored in real-time. Even better, task owners may get push-notifications before exceeding due dates or reaching standardization process limits and status meetings can be held without preparing a huge amount of slides, sheets and graphs, as all these data are continuously available.

2 Related Work

Approaching the simplicity in exchanging components at any time with the slogan "IT simply works" by Maragria and Steffen [1] is pointing in the direction of standards and frameworks to be used for product development and processes. The model, developed by Lee and Tang [4] identifies the economic outcome of the investment in standardization in a much simpler and not all-encompassing way, like aimed at with this thesis. The founder and Senior Advisor of the Lean Enterprise Institute, James P. Womack, is researching and transferring the way of Lean Thinking since 1991 in many industries.

3 Methods

Key to this development is the *one-thing approach* [2], which guarantees the continuous availability of one comprehensive model, the one thing, capturing all the required information. In particular it allows providing each stakeholder with a tailored view that focusses on the relevant information for the next step(s). This is important to easy the interaction in a way that all the involved parties can adequately participate, a precondition for the success of the introduction of a new technology [1].

4 Preliminary Results

Definition of an industrial standardization case as a proof of concept for the development of the above mentioned technology was accomplished. The domain modeling for the conceptual approach and gathering the requirements has started.

5 Next Steps

The variable variant modelling and product line development by Jörges, Lamprecht, Margaria, Schaefer and Steffen [3] describes a modelling framework with the approach towards standardization and product lining which will be used as the basic functionality for the development of this thesis. The domain model has to be enriched by the industrial standardization case requirements such as tailored view options, interdependencies of the products, the processes and the stakeholders and adding functions to enclose economic aspects.

References

1. Margaria, T., Steffen, B.: Simplicity as a Driver for Agile Innovation. IEEE Computer 43(6), 90–92 (2010)
2. Margaria, T., Steffen, B.: Agile IT: Thinking in User-Centric Models. In: Margaria, T., Steffen, B. (eds.) ISoLA 2008. CCIS, vol. 17, pp. 490–502. Springer, Heidelberg (2008)
3. Jörges, S., Lamprecht, A.-L., Margaria, T., Schaefer, I., Steffen, B.: A constraint-based variability modeling framework. STTT 14(5), 511–530 (2012)
4. Lee, H.L., Tang, C.S.: Modelling the Costs and Benefits of Delayed Product Differentiation. Management Science 44(1), 40–53 (1997)

Author Index