Alessandro Fantechi
Patrizio Pelliccione (Eds.)

LNCS 9274

Software Engineering for Resilient Systems

7th International Workshop, SERENE 2015
Paris, France, September 7–8, 2015
Proceedings

Springer

Lecture Notes in Computer Science **9274**

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

More information about this series at http://www.springer.com/series/7408

Alessandro Fantechi · Patrizio Pelliccione (Eds.)

Software Engineering for Resilient Systems

7th International Workshop, SERENE 2015 Paris, France, September 7–8, 2015 Proceedings

Springer

Editors
Alessandro Fantechi
University of Florence
Florence
Italy

Patrizio Pelliccione
University of Gothenburg
Gothenburgh
Sweden

ISSN 0302-9743 ISSN 1611-3349 (electronic)
Lecture Notes in Computer Science
ISBN 978-3-319-23128-0 ISBN 978-3-319-23129-7 (eBook)
DOI 10.1007/978-3-319-23129-7

Library of Congress Control Number: 2015946578

LNCS Sublibrary: SL2 – Programming and Software Engineering

Springer Cham Heidelberg New York Dordrecht London

Printed on acid-free paper

Springer International Publishing AG Switzerland is part of Springer Science+Business Media (www.springer.com)

Preface

The way software is developed is changing. It must take into account multifaceted constraints such as unpredictable markets, evolving customer requirements, pressures of shorter time-to-market, etc. At the same time, software is controlling critical functionalities in several domains such as transportation, health care, manufacturing, and IT infrastructures. As a result, modern software systems require on the one hand adding frequently (daily or weakly) new features, functionalities, or new versions of software artifacts according to changing contexts, business opportunities, or customer feedback, on the other hand ensuring their resilience – the ability of a system to persistently deliver its services in a dependable way even when facing changes, unforeseen failures, and intrusions.

The SERENE 2015 workshop provides a forum for researchers and practitioners to exchange ideas on advances in all areas relevant to software engineering for resilient systems, including, but not limited to:

1. **Development of resilient systems**
   - Incremental development processes for resilient systems
   - Requirements engineering and re-engineering for resilience
   - Frameworks, patterns, and software architectures for resilience
   - Engineering of self-healing autonomic systems
   - Design of trustworthy and intrusion-safe systems
   - Resilience at run-time (mechanisms, reasoning, and adaptation)
2. **Verification, validation, and evaluation of resilience**
   - Modelling and model-based analysis of resilience properties
   - Formal and semi-formal techniques for verification and validation
   - Experimental evaluations of resilient systems
   - Quantitative approaches to ensuring resilience
   - Resilience prediction
3. **Case studies and applications**
   - Empirical studies in the domain of resilient systems
   - Methodologies adopted in industrial contexts
   - Cloud computing and resilient service provisioning
   - Resilient cyber-physical systems and infrastructures
   - Global aspects of resilience engineering: education, training, and cooperation

This volume contains the papers presented at SERENE 2015, 7th International Workshop on Software Engineering for Resilient Systems, held during September 7–8, 2015 in Paris.

There were 18 submissions. Each submission was reviewed by at least three, and on average 3.1, Program Committee members. The committee decided to accept 10 papers.

We adopted EasyChair for managing the submission, reviewing, and proceedings phases.

July 2015

Patrizio Pelliccione
Alessandro Fantechi

Organization

Organizing Committee

Program Chairs

Patrizio Pelliccione	Chalmers University of Technology and University of Gothenburg, Sweden
Alessandro Fantechi	University of Florence, Italy

Publicity Chair

Laura Carnevali	University of Florence, Italy

Web Chair

Mirco Franzago	University of L'Aquila, Italy

Steering Committee

Didier Buchs	University of Geneva, Switzerland
Henry Muccini	University of L'Aquila, Italy
Patrizio Pelliccione	Chalmers University of Technology and University of Gothenburg, Sweden
Alexander Romanovsky	Newcastle University, UK
Elena Troubitsyna	Abo Akademi University, Finland

Program Committee

Marco Autili	Università dell'Aquila, Italy
Paris Avgeriou	University of Groningen, The Netherlands
Didier Buchs	CUI, University of Geneva, Switzerland
Barbora Buhnova	Masaryk University, Czech Republic
Andrea Ceccarelli	University of Florence, Italy
Ivica Crnkovic	Mälardalen University, Sweden
David De Andrés	Universidad Politécnica de Valencia, Spain
Vincenzo De Florio	Universiteit Antwerpen, Belgium
Felicita Di Giandomenico	ISTI-CNR, Italy
Alessandro Fantechi	University of Florence, Italy
Nikolaos Georgantas	Inria, France
Holger Giese	Hasso Plattner Institute for Software Systems Engineering, Germany
Nicolas Guelfi	University of Luxembourg
Mohamed Kaaniche	LAAS-CNRS

Vyacheslav Kharchenko	KhAI - Aerospace University, Kharkiv, Ukraine
Nuno Laranjeiro	University of Coimbra, Portugal
Istvan Majzik	Budapest University of Technology and Economics, Hungary
Paolo Masci	Queen Mary University of London, UK
Marina Mongiello	Politecnico di Bari, Italy
Henry Muccini	Università dell'Aquila, Italy
Sadaf Mustafiz	McGill University, Canada
András Pataricza	Budapest University of Technology and Economics, Hungary
Patrizio Pelliccione	Chalmers University of Technology and University of Gothenburg, Sweden
Markus Roggenbach	Swansea University, UK
Alexander Romanovsky	Newcastle University, UK
Stefano Russo	Università di Napoli Federico II
Peter Schneider-Kamp	University of Southern Denmark
Marco Vieira	DEI-CISUC, University of Coimbra, Portugal
Katinka Wolter	Freie Universitaet zu Berlin, Germany
Apostolos Zarras	University of Ioannina, Greece

Additional Reviewers

Capozucca, Alfredo
Cavezza, Davide
Dyck, Johannes
Frattini, Flavio
Lawrence, David
Manteuffel, Christian
Racordon, Dimitri
Ries, Benoit
Vogel, Thomas

Contents

Biological Immunity and Software Resilience: Two Faces of the Same Coin?

Marco Autili, Amleto Di Salle, Francesco Gallo(✉), Alexander Perucci, and Massimo Tivoli

Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica, Università dell'Aquila, L'Aquila, Italy
{marco.autili,amleto.disalle,francesco.gallo,massimo.tivoli}@univaq.it, alexander.perucci@graduate.univaq.it

Abstract. Biological systems modeling and simulation is an important stream of research for both biologists and computer scientists. On the one hand, biologists ask for systemic approaches to model biological systems to the purpose of simulating them on a computer and predicting their behavior, which is resilient by nature. This would limit as much as possible the number of experiments in laboratory, which are known to be expensive, often impracticable, hardly reproducible, and slow. On the other hand, beyond facing the development challenges related to the achievement of the resilience to be offered by biological system simulators, computer scientists ask for a well-established engineering methodology to systematically deal with the peculiarities of software resilient systems, in their more general sense. In line with this, in this paper we report on our preliminary study of immune systems (a particular kind of biological systems) aimed at devising software abstractions that enable the systematic modeling of resilient systems and their automated treatment. We propose a bio-inspired concept architecture for structuring resilient systems based on the Akka implementation of the widely-known Actor Model, which supports scalable and resilient concurrent computation. To the best of our knowledge, this work represents a first preliminary step towards devising a bio-inspired paradigm for engineering the development of resilient software systems.

1 Introduction

Biological systems modeling and simulation is an important stream of research for both biologists and computer scientists. As stated by Hofmeyr [16], the importance of adopting a systemic approach to biology is not new. Its relevance in the modern biological context comes from the need of robust mathematical models and computer simulations that faithfully predict the behaviour of entire biological systems, which are resilient by nature. In this direction, the construction of predictive models of bio-molecular networks is of paramount importance.

Based on differential equations, cellular networks of moderate size have been modeled successfully. However, when large-scale networks are concerned, the

A. Fantechi and P. Patrizio (Eds.): SERENE 2015, LNCS 9274, pp. 1–15, 2015.
DOI: 10.1007/978-3-319-23129-7_1

construction of predictive quantitative models is not easy, if not impossible, due to limited knowledge of mechanistic details and kinetic parameters. Indeed, the knowledge about these systems is typically available in the form of a textual description plus some informal diagrams, which often lead to ambiguities.

Furthermore, as noted by Sackmann et al. [27], the amount of biological knowledge is increasing, and experiments in laboratory tend to be expensive, slow, and often impracticable. As a result, the assistance of computers is becoming indispensable. Computer-assisted experiments could be less expensive, faster and more easily reproducible: "*executable software models of biological systems can be used for predictions, preparation and elimination of unnecessary, dangerous or unethical laboratory experiments*" [21]. As discussed in Sect. 6, one of the major questions systems biology is currently trying to answer is how to represent biological knowledge in a machine-processable way.

On the other side of the coin, beyond facing the development challenges related to the achievement of the resilience to be offered by biological system simulators, computer scientists ask for a well-established engineering methodology to systematically deal with the peculiarities of software resilient systems, in their more general sense [18]. By observing a strong analogy with biological systems, software engineering approaches in the literature (see Sect. 6) achieve resilience by means of mechanisms, e.g., replication, containment, isolation and delegation, which ensure that parts of the system can fail and recover without compromising the system as a whole. For example, recovery of components can be delegated to another (external) component and high-availability is ensured by replication where necessary. Thus, just like biological systems, software resilient systems are more flexible, loosely-coupled, scalable, and more amenable to change. They are significantly more tolerant of failure with respect to non-resilient systems. Still confirming the conceptual relation between biological systems and software resilient systems, biological immunity is related to the ability of an organism to resist a particular infection or toxin by the action of specific antibodies or sensitized white blood cells. This ability recalls the concept of software resilience, i.e., *the ability of a system to persistently deliver its services in a dependable way even when facing changes, unforeseen failures and intrusions.*

The work proposed in this paper starts from the observation that biological immunity and software resilience may be considered as two faces of the same coin. As a particular kind of biological systems, immune systems are resilient systems par excellence. Thus, while architecting resilient software systems, it does make sense to be inspired by the fundamental elements, relations, and behaviors of immune systems.

In line with the above, in this paper we report our preliminary study of immune systems aimed at devising software abstractions that enable systematic modeling and automated treatment of resilient software systems. We propose a bio-inspired concept architecture for structuring resilient systems based on the Akka[1] implementation of the widely-known Actor Model [15], which supports scalable and resilient concurrent computation. To achieve this, we have devised

[1] http://akka.io/.

an abstraction of immune system elements that are then mapped to concrete concepts of the Akka Actor Model. To the best of our knowledge, this work represents a first preliminary step towards devising a bio-inspired paradigm for engineering the development of resilient software systems.

The paper is organized as follows. In Sect. 2, we set the context of our work by summarizing the fundamentals of immune systems and introducing the devised abstractions. Leveraging these abstractions, in Sect. 3 we briefly describe three immune system scenarios that are representative with respect to the resilience concept. Basing on the Akka Actor Model introduced in Sect. 4, in Sect. 5, we propose a bio-inspired concept architecture for resilient software systems and we apply it to the described scenarios. Section 6 discusses related work, and Sect. 7 concludes and outlines future research directions that we will undertake to extend and put in practice the proposed concept architecture, and to precisely define the bio-inspired paradigm on top of it.

2 Immune Systems

An immune system [16] is a particular kind of biological system that is self-protecting against diseases. It is made of biological structures and processes within an organism. The minimum biological structure within an immune system is the cell, which in turn is made of molecules.

A key feature of an immune system is the ability to distinguish between (i) non-infectious structures, which must be preserved since they do not represent a disease, and (ii) infectious structures, i.e., *pathogens*, which must be removed since they result in injuries to the organism the immune system belongs to.

The discrimination between non-infectious and infectious structure takes place at the molecular level and is mediated by specific cell structures that enable the presentation and recognition of harmful components referred to as *antigens* (i.e., small fragments of a pathogen). In particular, these cell structures are able to detect anomalous/undesired situations through antigens recognition, and to place the immune system in a state of alarm. From this state, other cell structures are in charge of reacting with a defensive response, hence removing infectious structures.

By referring to [19], the main elements of an immune system can be summarized as follows:

- **Lymphocytes.** They are the cells of an immune system. For the purposes of this paper, it is enough to distinguish between *T Cells* and *B Cells*:
  - **T Cells.** They can be of two kinds, *T Helper* and *T Killer*. The former are cells responsible for preventing infections by managing and strengthening the immune responses enabled by the recognition of antigens. The latter are cells able to destroy certain tumor cells, viral-infected cells, and parasites. Furthermore, they are responsible for down-regulating immune responses, when needed.
  - **B Cells.** They are responsible for producing *antibodies* in response to foreign proteins of bacteria, viruses, and tumor cells.

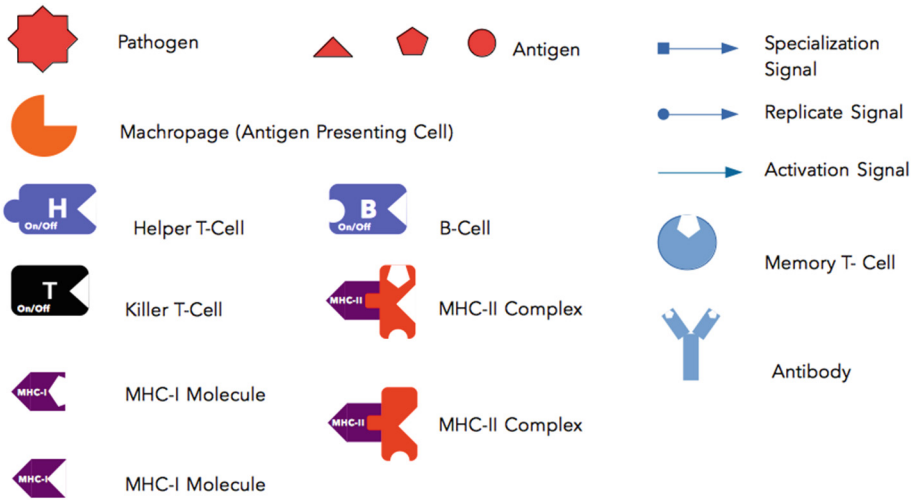


Fig. 1. Immune system components

Both B Cells and T Cells carry receptor molecules that make them able to recognize specific pathogens. In particular, T Killers recognize pathogens only after antigens have been processed and presented in combination with a *Major Histocompatibility Complex* (MHC) molecule. In contrast, B Cells recognize pathogens without any need for antigen processing.

- **Macrophages.** They are important in the regulation of immune responses. They are often referred to as *Antigen-Presenting Cells* (APC) because they pick up and ingest foreign materials and present these antigens to other cells of the immune system such as T Cells and B Cells. This is one of the important first steps in the initiation of an immune response.
- **Memory Cells.** When B Cell and T Cell are activated and replicated, some cells belonging to their progeny become long-lived *Memory Cells*. The role of Memory Cells is to build an immunological memory that makes the immune system stronger in being self-protecting to future infections/attacks. In particular, a Memory Cell remembers already recognized antigens and lead to a stronger immune response when these antigens are recognized again.
- **Immune response.** An immune response to foreign antigens requires the presence of APC in combination with B Cells or T Cells. When an APC presents an antigen to a B Cell, the B Cell produces antibodies that specifically bind to that antigen in order to kill/destroy it. If the APC presents an antigen to a T Cell, the T Cell becomes active. Active T Cells essentially proliferate and kill target cells that specifically express the antigen presented by the APC. The production of antibodies and the activity of T Killers are highly regulated by T Helpers. They send *signals* to T Killers in order to regulate their activation, proliferation (replication) and efficiency (specialization).

Following the description above, Fig. 1 shows the main elements of an immune system as constructs of a simple graphical notation that we use in Sect. 3 for immune system modeling purposes. The figure shows also the messages (signals) that the system elements can exchange.

3 Immune System Scenarios

By leveraging the graphical modeling notation introduced above, in this section, we briefly describe two scenarios in the immune systems domain, which are representative for a broad class of resilient systems. The scenarios provide the reader with a high-level description of the interactions that happen among the elements of an immune system during an immune response.

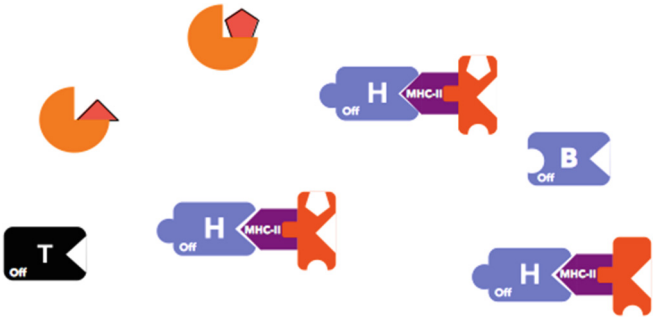


Fig. 2. Immune system - Scenario 0

Scenario 0. Figure 2 shows a scenario where the elements of an immune system are in an *inactive* state, marked with the *off* label. In particular, two APC engulf a virus or bacteria: each APC decomposes the pathogen (virus or bacteria) and exposes on its surface a piece of the pathogen, i.e., an antigen (see the triangle and pentagon in the figure). Metaphorically, we can think of this as a setup phase of a computer system, where each system's component is in an idle state and the antigen is a "perturbation" that comes from the outside or even by the system itself. In our context, we can see this perturbation as either a new or an anomalous, undesired, system behavior.

Scenario 1a and 1b. Continuing Scenario 0, the presence of an antigen causes the activation of one or more cells of the immune system (left-hand side of Fig. 3). In this case, the antigen is caught only by those cells that are able to treat it; so they switch from the *off* state to the *on* state. By referring to the right-hand side of Fig. 3, T Helpers recognize specific antigens and replicate themselves (replicate signals); T Helpers make T Killers and APC active (by sending to them an activation signal); T Killers replicate themselves. Some T Helpers specialize into Memory Cells (specialization signal). This scenario highlights how an immune response is a distributed and decentralized process.

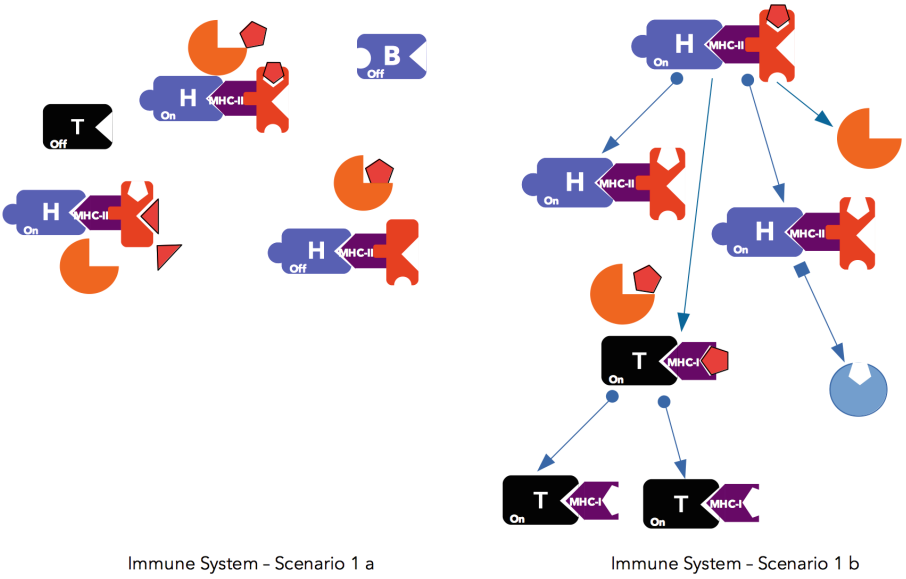


Fig. 3. Immune system - Activation scenarios

Scenario 1c. In this scenario, a B Cell becomes active after that it caught a known antigen and the T Helper close to it became active. This scenario points out that a B Cell has less constraints to satisfy with respect to T Cells, i.e., a B Cell does not need to interface with MHC molecules (Fig. 4).

These simple, yet representative scenarios, lead us to observe that, as a particular kind of resilient system in a specific domain:

- an immune system is composed of loosely-coupled elements, hence promoting modularity;
- the elements of an immune system can interact with each other by exchanging asynchronous messages, hence enhancing reliability;
- an immune system can react to unforeseen events, e.g., intrusions, being therefore fault tolerance;
- an immune system can foresee, e.g., dangerous situations through preventive recognition, hence achieving robustness;
- the elements of an immune system have the ability to adapt to context changes and recover from undesired situations, therefore showing flexibility and evolvability.

4 Actor Model

The Actor Model is a formal mathematical model of concurrent computation that was first proposed by C. Hewitt et al. in 1973 [15]. Over the years, this model has seen several programming languages employing the notion of actor,

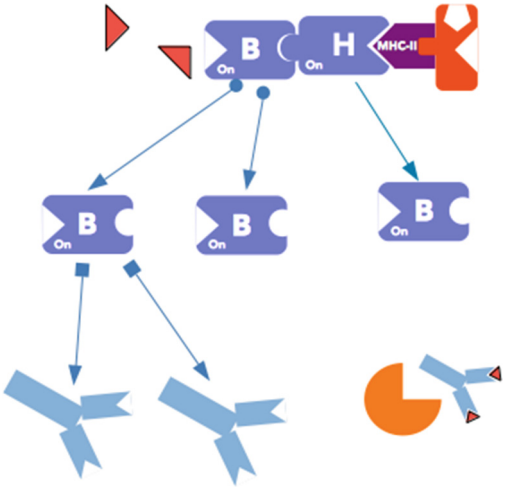


Fig. 4. Immune system - Scenario 1c

most notably Erlang [2] (a concurrent programming language designed for programming fault-tolerant distributed systems) and Scala [13] (which offers a concurrent programming model based on the Scala standard library dedicated to Actors).

For the purpose of this paper, we focus on the Akka toolkit[2], which is a framework that natively permits a programming style based on the Actor Model. Akka is written in Java and Scala, and offers an unified runtime and programming model that allows for building highly concurrent, distributed, and resilient message-driven applications.

In this section, after introducing the Actor Model, we present the Akka framework (Sect. 4.1), whose elements are then used in Sect. 5 to present our Akka-based Concept Architecture.

The Actor Model is characterized by (i) inherent concurrency of computation within and among Actors, (ii) dynamic creation/replication of Actors, (iii) inclusion of Actor addresses in messages, and (iv) interaction only through direct asynchronous message passing.

The Akka Actor Model achieve resilience by offering software abstractions that support the systematic development of flexible, loosely-coupled and scalable applications, which are significantly more tolerant of failure and stay responsive in the face of failure. Resilience is achieved through replication, containment, isolation and delegation mechanisms. Specifically, Akka components are isolated from each other, thereby ensuring failures isolation within the affected component only; the failing parts of the system have the possibility to recover without compromising the system as a whole; recovery actions are delegated to other

[2] http://akka.io.

(supervisor) components and, when needed, components replication is used to achieve high-availability.

4.1 Akka Actor Model

Actors are objects that encapsulate state and behavior, and communicate through message passing. Akka allows for modeling applications in terms of interacting actors that can be dynamically assigned sub-tasks, and arranging related functions into an organizational/hierarchical structure while thinking about how to tolerate/escalate failures.

An actor has its own state and can be seen as a container for Behavior, a Mailbox, Children and a Supervisor Strategy. Actors have a hierarchical structure and each actor has a supervisor, with the root supervisor being the `SystemActor`. In more details:

Actor Reference: actor references are used to represent actors to the outside. References enables transparency in the sense that an actor can be, e.g., restarted without needing to update references elsewhere, or seamlessly moved on remote hosts.

Behavior: messages are matched against the current behavior of the actor, i.e., to functions which defines the actions to be taken in reaction to the message. Importantly in our setting, the behavior of an actor can be changed dynamically, e.g., to allow the actor to come back to work after an "out-of-service" state is reached.

Mailbox: the mailbox connects the sender and receiver actor, and allows for enqueuing the exchanged messages in order to support asynchronous communication.

Children: an actor is potentially a supervisor and can create children for delegating sub-tasks. The creation and termination actions are not blocking (i.e., they happen behind the scenes in an asynchronous way).

Supervisor Strategy: a supervisor actor has strategy for handling faults of its children. For our purpose, the important aspect here is that fault handling is done transparently by the Akka framework, by exploiting monitors and by applying the available supervision strategies. Moreover, another crucial aspect towards achieving resilience is that strategies can be updated/added dynamically.

5 Akka-Based Concept Architecture

Figure 5 shows the bio-inspired concept architecture we are working on. It represents the starting point to support the bio-inspired paradigm that we have in mind for engineering the development of resilient software systems. Indeed, by referring to the scenarios 0, 1a, and 1b described in Sect. 3, we show only those elements that are strictly needed to provide the reader with intuitions on how to put together the elements of the concept architecture into a logically coherent argumentation.

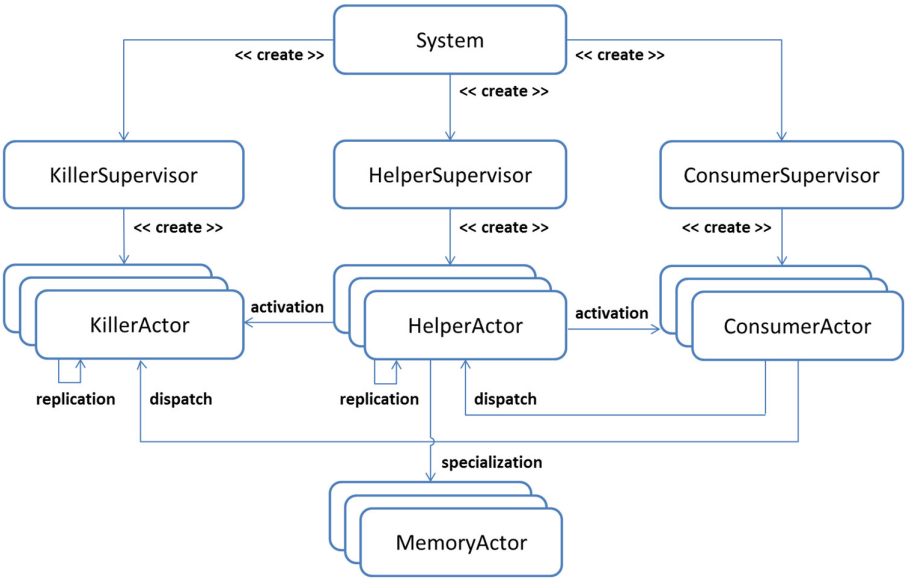


Fig. 5. Bio-inspired concept architecture for resilient systems

According to the Akka framework, the `System` actor creates the `KillerSupervisor`, `HelperSupervisor` and `ConsumerSupervisor` *supervisor* actors. By taking inspiration from immune systems (Sect. 2) and related scenarios (Sect. 3), the supervisor actors are in charge of detecting changes, intrusions, failures, and undesired behaviors. For instance, referring to systems where security is a crucial dependability requirement, the addition of a new component that behaves as a trojan, or the presence of a component that misbehaves. Another example, when referring to systems where performance is crucial, would be the presence of a component that does not perform as expected anymore.

Furthermore, supervisor actors are in charge of creating respective sub-actors, namely `KillerActor`, `HelperActor` and `ConsumerActor`, which are responsible for, e.g., self-protecting/self-reconfiguring the system from the detected undesired behavior/change by putting in place a resolutive response. This is done through recognition of the kind of problem and production/replication of those set of actors that can realize a suitable solution. E.g., removing, isolating, disabling the introduced trojan or the misbehaving component; replacing the component that badly perform with a new version of it that performs as expected.

In particular, by mimicking what happens in scenarios 0, 1a and 1b (Fig. 3) when the presence of an antigen causes the activation of one or more cells of the immune system and the antigen is caught only by those cells that are able to treat it, in our concept architecture, the occurrence of the problem/change is signalled by means of *dispatch messages*. Each message activates specific instances of `HelperActor` and `KillerActor`. The former play the role of res-

olutive response managers. After activation, a `HelperActor` instance aims at (i) inducing replication of `HelperActor` instances by sending *replication* messages; (ii) activating `KillerActor` and `ConsumerActor` instances by sending *activation* messages; and (iii) specializing some actor instances in `MemoryActor` instances by sending *specialization* messages. `KillerActor`s are actuators that consume messages and actually put in place the problem resolution or reconfiguration strategy that realizes the triggered response.

Considering resilience attributes of immune systems [3] and the support to resilience offered by the Akka Actor Model (Sect. 4), our concept architecture applied to the scenarios 0, 1a, and 1b is able to meet the following software resilience attributes:

- **agility**

  —*immune systems*: they have multiple barriers or layers of defense to prevent a pathogen from causing harm.
  —*software systems*: system undesired behaviors or changes represent the software counterpart of pathogens and our Actor Model implementation compartmentalizes the, e.g., undesired behavior, and allows its resolution locally without compromising the operation of the system as a whole.

- **redundancy**

  —*immune systems*: having the ability to replicate antibodies increases the probability that a pathogen that matches these antibodies will be stopped. Furthermore, generated antibodies keep memory of the already matched pathogens, hence strengthening the ability to stop it.
  —*software systems*: Akka toolkit offers persistence that enables actors to persist their internal state so that it can be recovered when a produced or replicated actor is started, restarted after a JVM crash or by a supervisor, or migrated in a cluster.

- **dynamic learning**

  —*immune systems*: as a redundancy enabler, based on previous knowledge about already recognized antigens, the immune system is able to learn new disturbances and related resolutive responses.
  —*software systems*: if some change or undesired behavior occurs at run time, and the affected actor is not able to manage it, the actor initially treats it as an unknown message. As a such, the message is passed to its supervisor. Based on historical knowledge, the supervisor is able to learn if there are other actors able to deal with the change or undesired behavior. If it is the case, the related resolutive logic is dynamically injected into the affected actor through the `PartialFunction` mechanism offered by Akka. Clearly, different policies can be applied when the supervisor has not been able to learn possible solutions, e.g., killing the affected actor, or requiring human intervention.

– **flexibility**

  −*immune systems*: antibodies are produced and added depending on the need, without any redesign.
  −*software systems*: the flexibility concept is native in Akka since actors can be dynamically produced or replicated, without blocking the system.

– **robustness**

  −*immune systems*: leveraging Apoptosis [23] or "programmed cell death" as a mechanism for deletion of "unwanted" cells, an immune system has the ability to keep working even when multiple cells are killed since not properly working anymore.
  −*software systems*: in the Actor Model, robustness is a form of fault tolerance. It uses the "let it crash" policy to manage the programmed death of faulty components that can be dynamically killed or stopped for preserving the system functioning, if possible, or at least for preventing dangerous system's misbehaviours.

6 Related Work

In this section, we discuss related work in the areas of (i) biological systems modeling and simulation and (ii) software resilient systems.

Biological systems modeling and simulation. Current approaches to biological systems modeling and simulation can be organized into three classes, namely *quantitative*, *qualitative*, and *rule-based* approaches.

Quantitative approaches [12,29]. They make use of differential equations and stochastic simulation to model biological processes. They essentially suffer two main issues. On the one hand they do not scale in the heterogeneity of constituent elements of the biological system, hence preventing their applicability to biological processes with many species and variables. On the other hand, as complex mathematical models, they are hard to be exploited and difficult to understand by end-users, not only biologists but also software engineers.

Qualitative approaches [21,27]. They are primarily based on the biological system's network structure and, differently from quantitative approaches, do not require knowledge about internal parameters of the system, e.g., kinetic parameters. Rather, models can be produced to abstract different views of a biological process hence allowing to reason at different biological organization levels, e.g., sub-cellular, cellular, tissue, organ, organism and ecosystem. The focus, here, is to identify how different components are connected together, how they are controlled and how they behave when functioning as a system. However, for these approaches to be effective and profitable, two main aspects must be ensured: (i) the model representation language must be rich enough to represent the various heterogeneous system's elements and to capture all the system behaviours

at the different organization levels; and (ii) the system identification technique must be powerful enough to identify substantially complex models, which can enable realistic simulation. The class of qualitative approaches comprises various formalisms that span from Boolean Network models [30] to constraint-based models [26], to Petri Nets [4,14], to logical models [20,24]. In addition to static analysis of the structural properties of a biological system's network topology, Petri Nets and logical models enable to reason on the systems behaviour by means of discrete dynamic modeling [1,8,24,30].

Rule-based approaches. They promote the production of rule-based models by using specialized languages such as BioNetGen language [9] or Kappa [6]. Being more reusable than equations, rule-based models allow for enhancing modularity, hence making tractable the modeling and analysis [11] of complex biological systems involving several different species [7]. Rules can also be used to generate simulations, both deterministic and agent-based [28].

Software resilient systems. The work in [17] proposes an approach to automatically detect faults of a redundant duplex system by using a model checker. The proposed method analyzes vulnerabilities of different variants of an algorithm by applying fault injection modelling. Relying on the Event-B model, in [22], the authors present a formal approach to model and assessing reconfigurable systems that guarantees resilience of data processing. The proposed system architecture is able to dynamically scale and reconfigure.

Natural and Biological systems, such as Ant colonies and Immune systems, have several features that can be exploited in designing and developing resilient systems. More precisely, these super organisms often use self-organizing behaviors and feedback loops [5] that allow the system to achieve reliable and robust solutions using information gathered from entities [25], without centralized control. The biological immune system can be seen as a massively distributed architecture: the multitude of independent cells work together resulting in the emergent behavior of the immune system. The immune system evolves to adapt and improve the overall system performance (e.g., organizational memory) [10,31]. These systems can be seen as complex collective systems in which the behaviour emerges from the product of interactions between individual entities. These entities follow a simple set of rules (i.e., not via top-down mechanism) and react only to their local environment. Features and principles as bottom-up mechanisms, feedback loops could be used for designing a scalable, adaptive and efficient framework.

7 Conclusions and Future Work

In this paper we proposed a bio-inspired concept architecture for resilient software systems based on the Akka Actor Model. Our proposal originates from the observation that immune systems natively enjoy resilience properties that have a direct counterpart in software resilient systems. Our long term goal is

to reach a mature enough knowledge about the key concepts that are common to both immune systems and resilient systems. By transposing (i) the elements of immune systems, (ii) their self-protecting behavior, and (iii) their relationships/interaction with other elements into their respective software design principles, this knowledge would allow us to elevate our preliminary concept architecture to a rigorously defined bio-inspired architectural style for resilient systems. This architectural style would constitute a common ground on top of which building the novel biological development paradigm we have in mind for software resilient systems.

To reach this goal, several challenges have to be faced. They are related to the definition and realization of general-purpose mechanisms that, being amenable of domain-specific customizations, support features such as:

- *automatic recognition* of software failures/changes through, e.g., run-time monitoring, feedback loops;
- *dynamic learning* of the solutions required to correctly react to the recognized failures/changes based on, e.g., dynamically acquired historical knowledge;
- *modular actuation* of the (learned) solution, without compromising the overall system function;
- *opportunistic selection* of those available solutions that better fit some predefined policy, e.g., to guarantee specified non-functional requirements;
- *self-stabilization* of the self-* actions, e.g., self-adaptation, self-reconfiguration, to guarantee the system equilibrium with respect to some specified invariant properties, despite continuous applications of self-* actions;
- *multilayer management* of failures/changes (and related strategies) in a modular, yet cohesive, way depending on the affected layer(s), e.g., application, middleware, operating system, network layer.

References

1. Bio-pepa: A framework for the modelling and analysis of biological systems. Theoretical Computer Science, 410(33–34), 3065–3084 (2009)
2. Armstrong, J.: Erlang. Commun. ACM **53**(9), 68–75 (2010)
3. Chandra, A.: Synergy between biology and systems resilience, master's thesis, missouri university of science and technology (2010)
4. Chaouiya, C.: Petri net modelling of biological networks. Briefings Bioinform. **8**(4), 210–219 (2007)
5. 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.) Software Engineering for Self-Adaptive Systems. LNCS, vol. 5525, pp. 1–26. Springer, Heidelberg (2009)
6. Danos, V., Feret, J., Fontana, W., Harmer, R., Krivine, J.: Rule-Based modelling, symmetries, refinements. In: Fisher, J. (ed.) FMSB 2008. LNCS (LNBI), vol. 5054, pp. 103–122. Springer, Heidelberg (2008)
7. Deeds, E.J., Krivine, J., Feret, J., Danos, V., Fontana, W.: Combinatorial complexity and compositional drift in protein interaction networks. PLoS ONE **7**(3), 03 (2012)

8. Dematté, L., Priami, C., Romanel, A.: The blenx language: a tutorial. In: Bernardo, M., Degano, P., Zavattaro, G. (eds.) SFM 2008. LNCS, vol. 5016, pp. 313–365. Springer, Heidelberg (2008)
9. Faeder, J., Blinov, M., Hlavacek, W.: Rule-based modeling of biochemical systems with bionetgen. In: Maly, I.V. (ed.) Systems Biology, volume 500 of Methods in Molecular Biology, pp. 113–167. Humana Press (2009)
10. Farmer, J., Packard, N.H., Perelson, A.S.: The immune system, adaptation, and machine learning. Physica D **22**(13), 187–204 (1986)
11. Feret, J., Danos, V., Krivine, J., Harmer, R., Fontana, W.: Internal coarse-graining of molecular systems. Proc. Nat. Acad. Sci. **106**(16), 6453–6458 (2009)
12. Gillespie, D.T.: Exact stochastic simulation of coupled chemical reactions. J. Phys. Chem. **81**(25), 2340–2361 (1977)
13. Haller, P., Odersky, M.: Scala actors: unifying thread-based and event-based programming. Theoret. Comput. Sci. **410**(2–3), 202–220 (2009)
14. Hardy, S., Robillard, P.N.: Pn: Modeling and simulation of molecular biology systems using petri nets: modeling goals of various approaches. J. Bioinform. Comput. Biol. **2–4**, 2004 (2004)
15. Hewitt, C., Bishop, P., Steiger, R.: A universal modular ACTOR formalism for artificial intelligence. In: Proceedings of the 3rd International Joint Conference on Artificial Intelligence. pp. 235–245. Standford, CA, August 1973
16. Hofmeyr, S.A.: An interpretative introduction to the immune system. In: Design Principles for the Immune System and Other Distributed Autonomous Systems, pp. 3–26. Oxford University Press (2000)
17. Höller, A., Kajtazovic, N., Preschern, C., Kreiner, C.: Formal fault tolerance analysis of algorithms for redundant systems in early design stages. In: Majzik, I., Vieira, M. (eds.) SERENE 2014. LNCS, vol. 8785, pp. 71–85. Springer, Heidelberg (2014)
18. Majzik, I., Vieira, M. (eds.): SERENE 2014. LNCS, vol. 8785. Springer, Heidelberg (2014)
19. Janeway Jr., C., Travers, P., Walport, M., et al.: Immunobiology: The Immune System in Health and Disease, 5th edn. Garland Science, USA (2013)
20. Klamt, S., Saez-Rodriguez, J., Lindquist, J.A., Simeoni, L., Gilles, E.D.: A methodology for the structural and functional analysis of signaling and regulatory networks. BMC Bioinform. **7**, 56 (2006)
21. Krepska, E., Bonzanni, N., Feenstra, A., Fokkink, W.J., Kielmann, T., Bal, H.E., Heringa, J.: Design issues for qualitative modelling of biological cells with petri nets. In: Fisher, J. (ed.) FMSB 2008. LNCS (LNBI), vol. 5054, pp. 48–62. Springer, Heidelberg (2008)
22. Laibinis, L., Klionskiy, D., Troubitsyna, E., Dorokhov, A., Lilius, J., Kupriyanov, M.: Modelling resilience of data processing capabilities of CPS. In: Majzik, I., Vieira, M. (eds.) SERENE 2014. LNCS, vol. 8785, pp. 55–70. Springer, Heidelberg (2014)
23. Lawen, A.: Apoptosisan introduction. BioEssays **25**(9), 888–896 (2003)
24. Morris, M.K., Saez-Rodriguez, J., Sorger, P.K., Lauffenburger, D.A.: Logic-based models for the analysis of cell signaling networks. Biochem. **49**(15), 3216–3224 (2010)
25. Nieh, J.C.: A negative feedback signal that is triggered by peril curbs honey bee recruitment. Curr. Biol. **20**(4), 310–315 (2010)
26. Papin, J.A., Palsson, B.O.: Topological analysis of mass-balanced signaling networks: a framework to obtain network properties including crosstalk. J. Theoret. Biol. **227**(2), 283–297 (2004)

27. Sackmann, A., Heiner, M., Koch, I.: Application of petri net based analysis techniques to signal transduction pathways, BMC Bioinform. **7**–**482** (2006)
28. Sneddon, M.W., Faeder, J.R., Emonet, T.: Efficient modeling, simulation and coarse-graining of biological complexity with NFsim. Nat. Meth. **8**(2), 177–183 (2011)
29. Srivastavawz, R., Youw, L., Summersy, J., Yin, J.: on stochastic vs. deterministic modeling of intracellular viral kinetics (2002)
30. Wang, R.-S., Saadatpour, A., Albert, R.: Boolean modeling in systems biology: an overview of methodology and applications. Phys. Biol. 9(5) (2012)
31. Watanabe, Y., Ishiguro, A., Shirai, Y., Uchikawa, Y.: Emergent construction of behavior arbitration mechanism based on the immune system. In: The 1998 IEEE International Conference on Evolutionary Computation Proceedings, IEEE World Congress on Computational Intelligence, pp. 481–486 (1998)

Towards Dynamic Software Diversity for Resilient Redundant Embedded Systems

Andrea Höller(✉), Tobias Rauter, Johannes Iber, and Christian Kreiner

Institute for Technical Informatics, Graz University of Technology, Graz, Austria
{andrea.hoeller,tobias.rauter,johannes.iber,christian.kreiner}@tugraz.at

Abstract. Faults in embedded systems are on the rise due to shrinking hardware feature sizes, increasing software complexity, and security vulnerabilities. Since such faults cannot be completely prevented, systems have to cope with their effects. Frequently, redundancy is used to achieve fault tolerance. However, with homogeneous redundancy common-cause faults such as software bugs or hardware faults in shared resources are not tolerated - diversity is needed.

In this paper, we highlight the potential of automatically introducing diversity via dynamic software diversity techniques. Recently, these techniques have attracted attention in the security domain. Furthermore, we present the idea of using such dynamic software diversity methods to create feedback-based systems that are able to adapt the execution of the program in such a way that the consequences of faults are leveraged. Finally, we demonstrate the approach with two use cases. We show that by using address space layout randomization - a widespread technique to prevent malicious attacks - it is possible to detect memory-related software bugs during runtime. Additionally, we illustrate the idea of adaptive dynamic software diversity by showing a simple example of how to recover from common-cause faults in the address decoder via software by inserting memory gaps with adjustable size.

Keywords: Software diversity · Self-adaptive systems · Resilience · Fault tolerance · ASLR · Redundancy

1 Introduction

Since ever more functionality is integrated into electronic devices, embedded systems have to fulfill increasing demands on high computing performance and have to realize ever more features [32]. At the same time, they have to offer dependability, since they are strongly integrated into everyday objects and play a crucial role in many critical application domains such as automotive, aerospace or critical infrastructures. Dependability is a superordinate concept regrouping different attributes such as safety, reliability, availability and security [2]. At the same time, embedded systems have to cope ever often with unforeseen scenarios caused by an increasing number of faults that jeopardize the dependability. The causes for this increasing number of faults are numerous:

A. Fantechi and P. Patrizio (Eds.): SERENE 2015, LNCS 9274, pp. 16–30, 2015.
DOI: 10.1007/978-3-319-23129-7_2

- *Random hardware faults*, such as soft errors and permanent errors due to manufacturing, process variations, aging, etc. occur more an more frequently due to shrinking hardware features sizes [28,34].
- *Software faults* are increasing due to growing software complexity [32].
- *Security attacks* pose an emerging risk, since ever more systems are interconnected [14].

To prevent such faults many systematic techniques for analyzing the dependability of systems have been proposed. For example, safety standards recommend failure mode and effects analysis, failure tree analysis or fault injection experiments [23]. These techniques are mainly used to assess the dependability of systems and to design appropriate countermeasures to prevent identified possible problematic situations in the presence of failures. However, the applicability of these approaches is limited, since they require detailed knowledge about the hard- and software system.

At the same time, the increasing demands on embedded systems lead to the trend to commercial off-the-shelf (COTS) hardware [27]. While there are processors available that are designed and produced for high reliability applications, their performance typically lags behind that of COTS multi-purpose processors. Additionally, COTS processors typically come at a much lower price than their reliability-hardened counterparts [13]. However, when using COTS-based hardware platforms, there are limited possibilities to add hardware-based fault tolerance techniques and details about the hardware (e.g. netlists, RTL models etc.) are typically not available.

Furthermore, the ever increasing complexity of embedded systems significantly complicates systematic approaches, since it is hard to identify all possible internal states and faults. To handle this challenge, we propose research towards the automated introduction of diversity in redundant systems as an alternative method to manage faults. This paper contributes towards this approach by

- presenting the idea of using the basic concepts of automated software diversity techniques that have been mainly proposed in the security domain to also increase the fault tolerance regarding non-malicious common-cause faults in redundancy-based embedded systems,
- introducing the concept of adaptive automated software diversity as a feedback-based method to react to observed malfunctions in order to establish resilience, and finally
- highlighting the potential of the approach by presenting two use cases showing that by using address space layout randomization - a widespread dynamic software diversity technique to prevent security gaps - it is possible to detect memory-related software bugs, and showing that adapting the size of dynamic memory gaps is a simple way of establishing resilience regarding memory address decoder faults.

2 Background and Related Work

2.1 Dependability and Resilience

System dependability attributes have a major impact on product development and product release as well as for company brand reputation. For this document we define dependability according to [2] as an integrating concept that encompasses the following attributes:

- *Safety*: the absence of catastrophic consequences on the users and environment.
- *Reliability*: the continuity of correct service.
- *Availability*: the readiness of correct service.
- *Security*: the concurrent existence of availability for authorized users only, confidentiality, and integrity with improper meaning of unauthorized.

To maintain safety despite faults, the system has to detect the malfunctioning in order to react in such a way that hazardous events leading to catastrophic consequences (e.g. injuries of people) are prevented. For example, a safety mechanism could switch into a safe state. To establish reliability and/or availability the consequences of the fault should be mitigated such that the system can continue the operation. Thus, resilience is needed. According to [10] software resilience refers to the robustness of a software to adapt itself so as to absorb and tolerate the consequences of failures, attacks or changes within and without the system boundaries. Consequently, to make systems resilient, they have to cope with changing circumstances regardless of their root cause. In order to deal with unforeseen events the idea of software self-adaptability has received attention [6]. For example, self-healing systems autonomously detect and recover from faulty states by changing their configuration. However, so far these techniques are mainly used in complex server systems.

Methods for embedded systems to recover from an unhealthy state are still a research challenge. Although hardware faults can be bypassed with self-modifying hardware (e.g. [24,33]), this technique is not applicable for COTS hardware and only offers limited flexibility. Thus, there remains the need for sophisticated software-based methods to handle unforeseen scenarios caused by faults.

2.2 Redundancy

Redundancy is a common way to establish fault tolerance [32]. As hardware is becoming ever cheaper due to Moore's law there are increasing opportunities to establish redundancy in a cost efficient way. For example, there is a rise of multicore technology in the embedded domain [26]. This allows to take advantage of the additional resources available in order to establish redundancy at a relatively low cost.

While spatial redundancy means that multiple program replicas that implement the same logical function are executed in multiple hardware channels, temporal redundancy techniques use only one hardware channel and perform the

same execution multiple times subsequently. Then, a voting scheme is used to detect faults by comparing the outputs of the redundant executions.

Typical redundancy techniques are *M-out-of-N* (MooN) architectures, where M channels out of a total N channels have to work correctly. For example, a Dual Modular Redundancy (*1oo2*) architecture means that there are two redundant channels, which are compared by a voter. If the outputs do not match, an error is detected. Depending on the application, the system can go into a safe state to prevent serious hazards. Since the voter is a single point of failure, it has to be highly reliable. Thus, it has to guarantee a high level of integrity (i.e., by using reliable hardware components and being certified with a high integrity level as described in safety standards). A *1oo2* system can guarantee data integrity as long as only one channel fails. However, if both variants are identical and include the same systematic fault, they fail in the same way and the fault is not detected. Thus diversity is needed to make it possible for the same fault to lead to different consequences. A classic approach to add diversity is N-version programming. This means that based on the same specification, several development teams work independently to design and implement N versions. Since this approach is very cost-intensive, we propose to automatically introduce diversity in the execution of the software.

The authors of [7] introduced a method combining adaptability and redundancy, by adapting the number N of redundant systems dynamically. Here, we propose not to adapt the number of redundant channels, but the behaviors of the channels.

2.3 Automated Software Diversity

Recently, automated diversity gained attention in the security domain as a technique of diversifying each deployed program version [25]. This forces attackers to target each system individually. A simple way of automatically introducing diversity in execution is to use different compilers and compiler options to generate multiple program versions. It has been shown that diverse compiling not only enhances the security of systems [37] but it can also increase the hardware and software fault tolerance [11,17]. In contrast to static techniques that generate multiple diverse program versions, dynamic software diversity techniques use only one binary that is deployed and introduces the diversity during operation. More details about automated software diversity research is provided in [4,25].

3 Dynamic Software Diversity Approach

Dynamic software diversity (DSD) techniques integrate randomization points in the executable. Examples of randomization points and their parameters are shown in Table 1. Then, the same program can perform diverse executions leading to the same results [4]. Diversity in execution can mean, for example, diverse performances, diverse memory locations, or diverse order of execution. Table 2 shows examples of dynamic diversity techniques. for example, data re-expression

Table 1. Examples of adjustable parameters of dynamic software diversity methods

Randomization method	Parameter
Memory gaps between objects [5]	Gap size
Changing base address of program [5]	Base address
Changing base address of libraries and stack [8]	Base address
Permutation of the order of routine calls variables [5]	Order of calls
Permutation of the order of variables [5]	Order of variables
Insertion of NOP instructions [22]	Number of NOPs
Data re-expression / data diversity	Parameter in re-expression
$(in = f(in,k), out = f^{-1}(out,k))$ [1]	algorithm (k)

Table 2. Classification of known automated dynamic diversity uses. While 'x' indicates the given goal is reported in literature, 'o' hints to possible goals for future research.

Known Use	*Level of Diversific.*			*Time of Diversific.*			*Goal*	
	Instruction	*Function*	*Program*	*Installation*	*Loading*	*Execution*	*Security*	*Reliability*
Data randomization [1]		x				x	o	x
Memory layout randomization (e.g. [5, 8])		x	x	x			x	o
Program encoding randomization (e.g. [3])	x					x	x	o
In-place diversification (e.g. [31])			x		x		x	o
Instruction location randomization (e.g. [16, 35]	x					x	x	o
Binary stirring [36]	x				x		x	o

is a well-established method to increase the fault tolerance with data diversity by transforming the original input to produce new inputs to redundant variants [1]. After execution the distortion introduced by the re-expression is removed before comparison. So a given initial data within the program failure region can be re-expressed to an input data that circumvents the faulty region [32].

However, most of the proposed DSD techniques focus on increasing the security by introducing uncertainty in the target. Today, if an attacker finds a vulnerability in a software program, he can exploit that knowledge to target all running copies of that program. Automated software diversity can decrease the software homogeneity and increase the cost to attackers by randomization implementation aspects of programs. The idea to diversify a software program each time it is deployed on a target recently gained attention in the security community [25].

The goal is to force the attacker to redesign the attack each time it is applied. Consequently, the risk of widely replicated attacks is reduced. For example, to circumvent buffer overflow attacks, used memory locations can be diversified. This forces the attacker to rewrite the attack code for each new target.

We propose to use such approaches also in redundant systems. If diverse replicas are used redundantly and their state is checked with a voting mechanism, an attacker has to find vulnerabilities in both replicas. Furthermore, we assume that it also increases the chance to detect non-malicious faults. The introduced diversity could lead to the detection of faults that have not been considered by systematic fault prevention techniques. For example, it is possible that programming bugs are detected that have not been found during the test and verification stage. Furthermore, hardware faults that appear during operation and affect all redundant executions can be identified, since the hardware is used differently by the diverse replicas. To sum it up, we propose to apply DSD techniques in redundant configurations to increase the chance to detect faults and to consequently enhance the safety.

4 Adaptive Dynamic Software Diversity Approach

A promising approach for resilience, is software that offers a reliable operation despite uncertain environments. Hardware faults, software bugs, or security exploits can be regarded as sources of uncertainty in the operation that has to be handled. For example, permanent hardware faults cannot be fixed during runtime. Thus, the software has to change the way it uses the faulty hardware such that the fault is masked. Adapting the software execution is probabilistic and does not require knowledge about the exact root cause of the fault. We propose to learn from detected anomalies and to diversify the execution with adaptive dynamic software diversity (ADSD). We define ADSD as

a method of automatically and dynamically diversifying the way that hardware and/or software components are used such that it learns from previously observed anomalies in order to increase the fault tolerance.

In [20], we provide a preliminary introduction of the idea of adaptive dynamic software diversity (ADSD) as a mean to bypass faults. Here, we provide further details of the approach and present a simple illustrative use case.

Figure 1 shows the basic structure of an ADSD system. Typically, a fault tolerant system contains the program, which performs the intended functionality of the system and a decision mechanism (DM) that monitors the program execution [32]. The DM detects anomalies, indicates alarms and decides which outputs to forward. For example, the diagnosis could be a plausibility check, a voter of a redundant system, or a self-aware technique that detects anomalies. Additionally, we propose a diversification control that creates a feedback-loop. This component manages the ADSD by collecting and analyzing data on detected anomalies obtained from the DM. The program is designed in such a way that it can be randomized during execution according to parameters that can be adjusted during runtime (see Table 1). Then, the diversification control can decide to alter

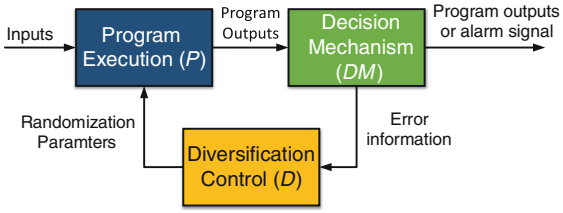


Fig. 1. Basic structure of ADSD. Based on information of a monitoring component (DM), a diversification controller decides whether and how to reconfigure the randomization mechanism of the main program [20].

the execution by changing one or multiple randomization parameters. Finally, the program reconfigures itself by using the adapted parameters.

5 Experimental Evaluation

We evaluated two use case applications in redundant configurations. We considered Dual Modular Redundancy (DMR) and Triple Modular Redundancy (TMR) for both use cases. The redundancy is established as temporal redundancy meaning that the replicas are executed one after the other. Each execution stores the corresponding result. Finally, a voter compares them after all replicas finished calculation.

A DMR system features two redundant channels. Such a system detects a fault, if the results of the redundant replicas are different. However, by purely comparing the outputs it is not possible to identify the faulty replica. This would require additional anomaly detection (i.e., plausibility checks). Here, we do not apply such mechanisms. Thus, we adapt the randomization mechanism of both replicas.

In a TMR configuration three replicas perform the same calculation. If all three outputs are different a warning is signaled and no output is forwarded, since the whole system seems to be corrupted. If at least two replicas provide the same output, this output is regarded to be correct and is forwarded. If two outputs match and one output is different, it is assumed that a fault occurred in the replica that produced the different output. A replica is also considered to contain a fault if an execution leads to a fail-stop failure. These failures can be detected relatively easily, since they lead to an observable crash of the system (e.g. segmentation fault or an infinite loop). If one replica is regarded as faulty, the randomization parameter of this replica is adapted. If it is observed that all three replicas lead to different outputs, all three replicas are reconfigured. For demonstration purposes, we change the reconfiguration parameter randomly. However, in order to achieve a more effective mechanism we plan to refine this approach to a more sophisticated logic in future work.

We evaluated how many faults can be detected by using the coverage metric

$$c_{detection} = \frac{\#detected\ faults}{\#faults}, \tag{1}$$

where $\#faults$ denotes the number of injected faults that actually have an impact on the reliability of the application, since they are not masked.

The techniques can not only be used to detect faults, but also to recover from them. To evaluate the ability to recover, we define the following metric:

$$c_{recovery} = \frac{\#bypassed\ faults}{\#faults}, \tag{2}$$

where $\#bypassed\ faults$ is the number of faults that could be mitigated with the evaluated technique.

5.1 Use Case I: Detecting Memory-Related Software Bugs via Address Space Layout Randomization

Here, we illustrate the potential of DSD techniques to increase the fault detection capabilities by evaluating the potential of the widely-established DSD technique address space layout randomization (ASLR) to detect memory-related software bugs during runtime.

Address Space Layout Randomization. ASLR randomizes base addresses to protect from security attacks [12]. It randomly arranges the starting address of the executable and the positions of the stack, heap, and libraries. To support ASLR the application has to be compiled in such a way that position independent code is generated. ASLR complicates memory corruption, code injection and code reuse attacks. For example, it is an effective countermeasure against buffer overflow attacks and it can prevent an attacker from jumping to a particular exploited function in memory. Today, ASLR is the only widely-deployed probabilistic defense against security attacks. It is supported by nearly all commonly used operating systems such as Linux (since kernel version 2.6.12), Android (since version 4.0), OS X (since version 10.5) and Microsoft Windows (since Windows Vista). However, as far as the authors of this work know, it has not been evaluated for safety or reliability purposes.

Evaluation Methodology. To assess the efficiency of fault tolerance mechanisms, a software including faults is required. We proceeded as follows to create benchmarks representing typical software faults:

- First, we used the GSM application of the MiBench benchmark suite for telecommunication applications [15] as a stating point.
- Then, we injected the most frequent types of software faults found in field studies.
- Finally, in order to further increase the representativity of the tested faults, we filtered out those faults that should be detected through software testing.

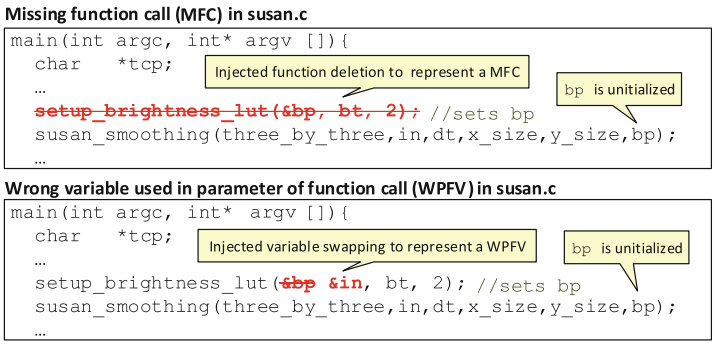


Fig. 2. Examples of injected memory-related Mandelbugs that are not detected by considered fault prevention mechanisms.

Software-Fault Injection. In order to represent realistic faults, we used the SAFE software-fault injection tool presented in [29]. It deliberately introduces faults into a software in order to assess its behavior in the presence of faults. The faults are injected by small changes in the program code. Various versions of a program are created, where each version includes another fault as exemplified in Fig. 2. This technique is similar to the well-known mutation test technique. However, while the goal of mutation testing is to evaluate the test suite, software-fault injection is meant to assess fault tolerance techniques.

The main purpose of software-fault injection is to represent residual faults. These are those faults that are not detected by rigorous design and testing and affect the safe operation of the system. For the development of the SAFE tool several field data were analyzed to characterize faults that can realistically occur in complex software [30]. The majority of bugs belong to a relatively small set of fault types and are independent from the particular system. More precisely, the SAFE tool injects a set of fault types that represent a total of about 68 % collected faults that where collected from real-world software. Table 3 shows the faults injected from the SAFE tool in our examples.

Table 3. Overview of injected fault types [29]

Fault type	Description
MFC	Missing function call
MIFS	Missing IF construct and statements
MVAE	Missing variable assignment using an expression
MVAV	Missing variable assignment using a value
MVIV	Missing variable initialization using a value
WPFV	Wrong variable used in parameter of function call

Table 4. Overview of generated test cases for the GSM benchmark

Number of injected fault types							Fault prevention			Bug classification	
MFC	MIFS	MVAE	MVAV	MVIV	WPFV	Total	Warning	Static A.	Und.	Bohrb.	Mandelb.
51	53	65	0	21	352	**542**	51	62	**426**	184	**247**

The SAFE tool assures that the source code including the injected fault is syntactically correct. In order to better reproduce the fault types observed in practice, additional constraints are applied to select fault locations. For instance, the MFC fault type only affects function calls that do not return any value or do not use the return value. Another example is that an *if*-construct is only removed (MIFS) in case the construct contains less than six statements. A programmer would likely notice a missing larger *if*-construct.

Filtering of Faults. In order to further increase the representativity of the tested faults, we do not consider faults that result in a warning of the compiler (option flags `-Wall -Wextra`). Furthermore, we filter out those faults that are found using the Clang static source code analyzer [9]. This drastically reduces the number of test cases (see Table 4).

The difficulty of detecting a software fault using testing depends on the determinism of its caused failure [29]. If a fault causes an error that always leads to the same failure for a given input, the fault can be detected relatively easily during testing. These kind of bug are called Bohrbugs. Another kind of bugs are Mandelbugs, which are much more difficult to detect, since they do not always lead to an error. Typically, the occurrence of Mandelbugs depends on timing (e.g. race conditions) or on the use of resources (e.g. addressing wrong memory regions). In this paper, we consider Mandelbugs that are memory-related. As shown in Table 4 this classification further reduces the number of evaluated test cases. To sum up, we generated 242 faults that are particularly hard to detect by testing for our evaluation.

ASLR Configurations. The binaries were generated with GCC v4.8.2 and executed natively with Ubuntu running on a PC featuring a single-core Intel Xeon CPU. In Linux the ASLR can be configured by a parameter called `randomize_va_space`. This parameter can be changed in order to set following options:

- `0`: Disable ASLR
- `1`: Randomize the base addresses of the stack, virtual dynamic shared object (VDSO) page, and the shared memory regions.
- `2`: Randomize the base addresses of the stack, virtual dynamic shared object (VDSO) page, the shared memory regions, and the data segment.

Experimental Results. We executed each binary containing one bug ten times for each ASLR configuration and stored the resulting output and applied this

Table 5. Detection coverage as defined of exemplary ASLR configurations regarding different programming mistakes leading to memory-related software-bugs.

Fault type		DMR configurations				TMR
Name	#Faults	0 vs.1	0 vs. 2	1 vs. 2	2 vs. 2	1 vs. 2 vs. 3
MFC	18	83 %	83 %	83 %	83 %	83 %
MIFS	15	0 %	0 %	0 %	0 %	0 %
MVAE	26	35 %	38 %	23 %	23 %	38 %
MVIV	8	38 %	38 %	13 %	13 %	38 %
WPFV	180	47 %	47 %	41 %	41 %	47 %
Total	**247**	**45 %**	**45 %**	**38 %**	**39 %**	**45 %**

procedure for the two input samples provided by the MiBench benchmark suite. The bugs never lead to a crash of the whole program, but always lead to a silent corruption of the output value.

Unfortunately, using ASLR it was not possible change the configuration of the execution in such a way that any of the injected software bug was mitigated and correct results were produced. However, frequently the erroneous outputs of diverse replicas were different leading to a high fault detection rate as shown in Table 5. Even if using the same configuration in a DMR configuration (2 vs. 2) on both redundant replicas in average about 39 % of all tested faults can be detected. When deactivating the ASLR feature on one channel and using it on the other channel (0 vs. 1 and 0 vs. 2) the detection coverage is even as high as 47 % for the tested application. The results also show that there is only a small gain of adding a third channel to establish a TMR system. The results indicate that ASLR is not only an effective method to prevent malicious attacks, but also enhances the fault detection of memory-related software bugs in redundant configurations. However, additional exhaustive experiments including more faults, benchmark applications, and input values, are required in order to make more reliable statements.

5.2 Use Case II: Adaption of Memory Gaps to Bypass Address Decoder Faults

Dynamic Software Diversity with Memory Gaps. To realize an automated software diversity mechanism we used random memory gaps as proposed in [5]. We arranged important variables used to control loops (`i`,`j`), storing the result (`n`), and manipulating the input in an early processing stage (`seed`) in a struct as shown in Fig. 3. Dummy variables with adjustable size are introduced in order to offer the possibility to manipulate the starting addresses of the protected variables. The size of the memory gaps represents the reconfiguration parameter that can be changed by the diversification control. Thus, if the diversification control decides to change a replica, the size of the memory gaps is adapted. We have chosen a random size between 0 and 64 bytes. Initially, the

```
int* bitcnt(int *results, int iterations, int gapsize){
  struct ProtVars {
    int dummy1[gapsize]; int  i;
    int dummy2[gapsize]; long j;
    int dummy3[gapsize]; long n;
    int dummy4[gapsize]; long seed;
  } protvars;
...
}
```

Fig. 3. Important variables are defined in a struct that introduces memory gaps between the variables with configurable size.

size of the memory gaps is configured differently for the three replicas (0 bytes, 32 bytes and 64 bytes). We plan further research to analyze the trade-off between a high degree-of freedom and additional memory requirements.

If no fault is detected, there is no need to diversify the program replicas. However, if there is a fault, the diversification control changes the size of the introduced memory gap. These is applied to all replicas when considering a DMR system or a TMR system where all three outputs are different. If only one output differs in a TMR configuration, only the erroneous replica is adapted.

Evaluation Methodology. Our exemplary use case is based on an implementation written in C and compiled for the ARM9 architecture. The application is the bitcount benchmark obtained from the MiBench benchmark suite for automotive applications [15]. The application counts the number of bits in an array of integers.

Fault Injection Framework. We simulated the benchmark application with a QEMU-based fault injection framework as presented in [18,19,21]. This framework allows to inject transient and permanent CPU and memory faults. We used it to inject permanent stuck-at address decoder faults. To model such faults the framework changes the address of the victim memory cells accordingly whenever it is accessed. This leads to accessing wrong memory locations.

Fault Injection Experiments. We injected 256 permanent stuck-at-1 and stuck-at-0 faults in the RAM addresses corresponding to the variables shown in Fig. 3. Due to masking effects, only 131 of these faults had an impact on the output values. Here, the injected fault lead to the situation that a wrong memory cell containing another value that the intended memory cell was addressed.

5.3 Experimental Results

As shown in Table 6 the homogeneous temporal redundant approach without the ADSD technique, failed to detect the introduced faults. The reason for this is that only one hardware channels was used and the fault in this hardware affected all

Table 6. Summary of the experimental results of use case II

	w/o ADSD	with adaptive memory gaps	
		DMR	TMR
Faults detected ($C_{coverage}$)	0 %	100 %	100 %
Faults tolerated ($C_{recovery}$)	0 %	82 %	94 %

executions in the same way, since they used exactly the same hardware resources. However, if the replicas use the memory slightly differently realized by a different initial configuration of the size of the memory gaps, the injected address decoder faults always lead to different consequences and thus are always detected.

Furthermore, if an error is detected the gapsizes are adjusted. For both redundant configurations - DMR and TMR - it was possible to recover from most of the detected faults. The TMR configuration performs better, since here it is possible to identify the faulty replica and thus only this replica is reconfigured. In summary, the results indicate that this adaptive technique is a simple and effective solution to recover from permanent address-decoder faults.

6 Conclusion

For many application domains of embedded systems dependability it is of utmost importance. At the same time the number of faults that may appear and the complexity of embedded systems increases dramatically, which hampers the application of traditional systematic approaches to prevent faults. Thus, methods are required that are able to detect unforeseen faults. To achieve this, we propose to use redundant systems that exhaustively integrate diversity in an automated way. In order to contribute towards this research direction, we proposed to exploit DSD techniques proposed in the security domain. The potential of this approach has been highlighted with a use case showing that using address space layout randomization allows to detect memory-related software bugs. Furthermore, we proposed to use DSD in a feedback-based configuration in order to establish resilience and exemplified this approach with a simple use case.

Although, the presented experiments indicate the potential of the proposed approach, more analysis is required in order to provide a stronger evidence. Thus, in the future, we plan to conduct more sophisticated and exhaustive evaluation of the proposed approaches. Furthermore, the actual realization of (A)DSD depends on the application and the considered fault model. Thus, we plan to develop techniques for specific complex applications and we hope to encourage further researchers to explore techniques based on the promising yet challenging idea of (A)DSD.

References

1. Ammann, P.P.E., Knight, J.C., Amman, P., Kngiht, J.: Data diversity: an approach to software fault tolerance. IEEE Trans. Comput. **37**(4), 418–425 (1988)

2. Avizienis, A., Laprie, J.C.J., Randell, B., Landwehr, C., Member, S.: Basic concepts and taxonomy of dependable and secure computing. IEEE Trans. Dependable Secure Comput. **1**(1), 11–33 (2004)
3. Barrantes, E.G., Ackley, D.H., Forrest, S., Stefanović, D.: Randomized instruction set emulation. ACM Trans. Inf. Syst. Secur. **8**(1), 3–40 (2005)
4. Baudry, B., Monperrus, M.: The multiple facets of software diversity: recent developments in year 2000 and beyond, ArXiv e-prints (2014)
5. Bhatkar, S., DuVarney, D., Sekar, R.: Efficient techniques for comprehensive protection from memory error exploits. In: USENIX Security Symposium (2005)
6. Brun, Y., Serugendo, G.D.M., Gacek, C., Giese, H., Kienle, H., Litoiu, M., Müller, H., Pezze, M., Shaw, M.: Software Engineering for Self-Adaptive Systems. Engineering self-adaptive systems through feedback loops. Springer, Heidelberg (2009)
7. Buys, J., De Florio, V., Blondia, C.: Towards context-aware adaptive fault tolerance in SOA applications. In: ACM International Conference on Distributed Event-Based System (2011)
8. Chew, M., Song, D.: Mitigating buffer overflows by operating system randomization. Technical Report CMUCS-02-197, Carnegie Mellon University (2002)
9. Clang Project: Clang Static Analyzer (2014)
10. Florio, V.D.: On resilient behaviors in computational systems and environments. J. Reliable Intell. Environ. **1**(1), 1–14 (2015)
11. Gaiswinkler, G., Gerstinger, A.: Automated software diversity for hardware fault detection. In: IEEE Conference on Emerging Technologies and Factory Automation (2009)
12. Giuffrida, C., Kuijsten, A., Tanenbaum, A.: Enhanced operating system security through efficient and fine-grained address space randomization. In: USENIX Conference on Security (2012)
13. Goloubeva, O., Rebaudengo, M., Reorda, M.M.S., Violante, M.: Software-Implemented Hardware Fault Tolerance. Springer, Heidelberg (2006)
14. Gubbi, J., Buyya, R., Marusic, S., Palaniswami, M.: Internet of things (IoT): a vision, architectural elements, and future directions. Future Gener. Comput. Syst. **29**, 1645–1660 (2013)
15. Guthaus, M.R., Ringenberg, J.S., Ernst, D., Austin, T.M., Mudge, T., Brown, R.B.: MiBench: a free commercially representative embedded benchmark suite. In: IEEE International Workshop on Workload Characterization (2001)
16. Hiser, J., Nguyen-Tuong, A., Co, M., Hall, M., Davidson, J.W.: ILR: where'd my gadgets go? In: IEEE Symposium on Security and Privacy (2012)
17. Höller, A., Kajtazovic, N., Römer, K., Kreiner, C.: Evaluation of diverse compiling for software fault tolerance. In: Design Automation and Test in Europe (2015)
18. Höller, A., Krieg, A., Rauter, T., Iber, J., Kreiner, C.: QEMU-based fault injection for a system-level analysis of software countermeasures against fault attacks. In: Euromicro Conference on Digital System Design2 (2015)
19. Höller, A., Macher, G., Rauter, T., Iber, J., Kreiner, C.: A virtual fault injection framework for reliability-aware software development. In: IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (2015)
20. Höller, A., Rauter, T., Iber, J., Kreiner, C.: Adaptive dynamic software diversity: towards feedback-based resilience. In: IEEE/IFIP International Conference on Dependable Systems and Networks - Supplementary Volume (2015)
21. Höller, A., Schönfelder, G., Kajtazovic, N., Kreiner, C.: FIES: a fault injection framework for the evaluation of self-tests for COTS-based safety-critical systems. In: IEEE Microprocessor Test and Verification Workshop (2014)

22. Homescu, A., Neisius, S., Larsen, P., Brunthaler, S., Franz, M.: Profile-guided automated software diversity. In: IEEE/ACM International Symposium on Code Generation and Optimization (2013)
23. ISO 26262: Road Vehicles - Functional Safety Standard (2009)
24. Jafri, S., Piestrak, S.J., Sentieys, O., Pillement, S.: Design of a fault-tolerant coarse-grained reconfigurable architecture : a case study. In: International Symposium on Quality Electronic Design (2010)
25. Larsen, P., Homescu, A., Brunthaler, S., Franz, M.: SoK: automated software diversity. In: IEEE Security and Privacy Magazine (2014)
26. Macher, G., Höller, A., Armengaud, E., Kreiner, C.: Automotive embedded software: migration challenges to multi-core computing platforms. In: IEEE Conference on Industrial Informatics (2015)
27. Madeira, H., Some, R.R., Moreira, F., Costa, D., Rennels, D.: Experimental evaluation of a COTS system for space applications. In: Proceedings of the 2002 International Conference on Dependable Systems and Networks, pp. 325–330 (2002)
28. Meza, J., Wu, Q., Kumar, S., Mutlu, O.: Revising memory errors in large-scale production data centers: analysis and modelling of new trends from the field. In: IEEE/IFIP International Conference on Dependable Systems and Networks (2015)
29. Natella, R.: Achieving representative faultloads in software fault injection. Ph.D. thesis. Università degli Studi di Napoli Federico II (2011)
30. Natella, R., Cotroneo, D., Duraes, J.A., Henrique, S.: On fault representativeness of software fault injection. IEEE Trans. Softw. Eng. **39**(1), 80–96 (2011)
31. Pappas, V., Polychronakis, M., Keromytis, A.D.: Smashing the gadgets: hindering return-oriented programming using in-place code randomization. In: IEEE Symposium on Security and Privacy (2012)
32. Pullum, L.: Software Fault Tolerance Techniques and Implementation. Springer, Heidelberg (2001)
33. Boesen, M.R., Pascal, S., Madsen, J.: Feasibility study of a self-healing hardware platform. In: Reconfigurable Computing: Architectures, Tools and Applications (2010)
34. Saggese, G.P., Wang, N.J., Kalbarczyk, Z.T.: An experimental study of soft errors in microprocessors. IEEE Micro **25**(6), 30–39 (2005)
35. Scott, K., Kumar, N., Velusamy, S., Childers, B., Davidson, J., Soffa, M.: Retargetable and reconfigurable software dynamic translation. In: International Symposium on Code Generation and Optimization (2003)
36. Wartell, R., Mohan, V., Hamlen, K.W., Lin, Z., Rd, W.C.: Binary stirring: self-randomizing instruction addresses of legacy x86 binary code. In: ACM Conference on Computer and Communications Security (2012)
37. Wheeler, D.A.: Fully countering trusting trust through diverse double-compiling. Ph.D. thesis. George Mason University (2009)

A Decomposition Method for the Verification of a Real-Time Safety-Critical Protocol

Tamás Tóth, András Vörös(✉), and István Majzik

Department of Measurement and Information Systems,
Budapest University of Technology and Economics, Budapest, Hungary
{totht,vori,majzik}@mit.bme.hu

Abstract. Formal methods, especially model checking techniques, are often used for the verification of the resilience of safety critical systems. The usual complexity of the verification problem in real life systems (due to state space explosion and the handling of time dependent behavior) demands efficient techniques. In this paper we propose a decomposition approach: the layered structure of the system is exploited to decompose the verification problem to smaller and tractable ones. In addition, the structure of the requirements (formalized as the combination of reachability and liveness properties) is also exploited to construct simpler verification problems for the model checker. The decomposition approach is demonstrated in case of the verification of a distributed protocol in a SCADA system that shall provide functionality even after the occurrence of a finite number of transient faults.

1 Introduction

The application of formal methods in the model based development and verification of safety critical systems is becoming a more and more important technique. Model checking is an automatic formal verification method for deciding if the model fulfills properties formalized in temporal logic. However, even for simple systems the complexity of verification might grow too large to be manageable due to the inherent distributed and timed characteristic of such systems. In particular, the verification of distributed systems often leads to the well-known phenomena of state space explosion which is a major obstacle for successful model checking. Real-time systems require methods being able to handle timed behaviors expressed with real-valued clock variables and their relations, further increasing the complexity of verification. Due to the above mentioned reasons, model checking techniques are often unable to verify complex real-time systems in a fully automatic manner. Decomposition can serve as a solution: safety critical systems are mainly composed hierarchically, where different layers of functions rely on each other. Experts can exploit this layered structure to decompose the verification problem to smaller and tractable ones. In addition, the specified properties in real-life systems are typically complex in the sense that they are usually combinations of reachability and liveness queries. On the basis of the expected behavior of the system and the structure of the property specification,

A. Fantechi and P. Patrizio (Eds.): SERENE 2015, LNCS 9274, pp. 31–45, 2015.
DOI: 10.1007/978-3-319-23129-7_3

experts can decompose the specification and give simpler verification problems to the model checker.

In this paper, this decomposition approach is demonstrated by the verification of a distributed safety critical protocol, whose main functionality is to guarantee reliable communication between components in a distributed SCADA (Supervisory Control and Data Acquisition) system. The protocol is hierarchically layered in the sense that it implements two functionalities: master election and the allocation of communication identifiers, where the latter functionality is based on the former one, i.e., performed by an elected master. The requirement for the protocol is to provide this functionality even after the occurrence of a finite number of transient faults. This requirement is formalized in linear temporal logic and a decomposition scheme is introduced in order to make verification feasible. The main goal of our work is to show how the structure of the system and the specification can be exploited to provide efficient verification. This decomposition approach is a generic scheme that can be followed in similar systems where the functions can be decomposed and a similar combination of reachability and liveness properties shall be verified.

The structure of the paper is the following. In Sect. 2, as the context of our work, the safety critical protocol is introduced. The background of our work is given in Sect. 3. The abstraction and decomposition rules are defined in Sect. 4, and then applied to the verification problem in Sect. 5. Finally, related work is mentioned and the conclusions of the paper are drawn.

2 Description of the Protocol

In this section, as the context and motivation of our work, the protocol and specified properties are introduced in details. The main purpose of the protocol is to ensure stable and fault tolerant communication between components of a distributed SCADA system. In the protocol, communication is performed in two layers: the lower layer serves for administration, while the upper layer transmits information between the components.

There are two types of components in the system: at most four communication units called as *eth*s, and at most ten input-output units called as *lio*s that are connected via a CAN bus that serves as the communication channel. Each component has a 29 bit physical address called *hwid* that is used in administrative messages to identify a specific component on the bus. However, components also get assigned a 4 bit logical address called *cid* that is used in the higher level communication protocols instead of *hwid* to save bandwidth. The *cid*s of *eth*s are assigned statically from the range [0...3], while *lio*s obtain their *cid* values dynamically from the range [4...13] from a distinguished *eth* that is an elected master. *cid* values 14 and 15 are reserved for addressing multicast and broadcast messages, respectively.

The functionalities of the protocol can be summarized as follows:

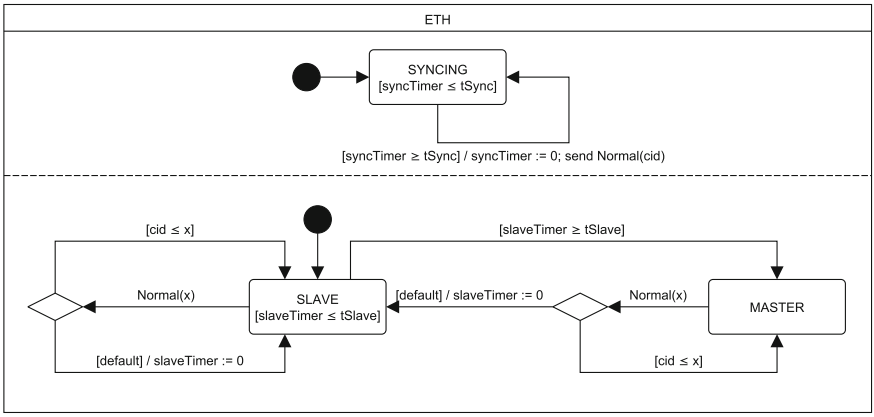


Fig. 1. Master election

– Master election. From the *eth*s that communicate on the bus, the one with the lowest *cid* value must be elected as master.
– Assignment of logical addresses. The master *eth* must ensure that all *lio*s have a unique *cid*.

Since the system is used in a critical context, it must provide the above functionalities even in the presence of a finite number of predefined faults. Accordingly, the verification must be aimed at the checking of the correct functionality of the protocol in a fault-free case and also in the presence of these faults. As the protocol was designed using SysML models (with time extensions), we will refer to the relevant statechart models to present the operation of the protocol. These statecharts were used to derive the formal models that were the basis of verification using our fault modeling and decomposition approach.

2.1 Master Election

To ensure that *lio*s obtain unique logical addresses, *cid*s can only be assigned by a distinguished *eth* called master. The purpose of master election is to ensure that during the operation of the system, the *eth* with the lowest *cid* is consistently considered as master by all *eth*s that are up. A simple timed statechart model of master election is depicted in Fig. 1.

The behavior of *eth*s defined by the statechart can be summarized as follows. Note that *syncTimer* and *slaveTimer* are clock variables that are used to define time dependent behavior (in the same way as clock variables are used in the common timed automata formalism [1]): their values are constantly increasing by an equal rate and can be checked in guard expressions and reset by actions. Moreover, state invariants can be defined (written into the state symbol in square brackets) that may also contain clock variables.

– A message *Normal*(*cid*) is broadcasted at every *tSync* time units with the *cid* of the *eth* as payload. This message serves as a heartbeat between *eth*s.
– Initially, the *eth* is a slave. If for the last *tSlave* time units the *eth* has not received any *Normal* messages with lower *cid* value than the *eth* itself has, then the *eth* becomes master.
– An *eth* remains master as long as it does not receive a message *Normal* with a *cid* lower than its own *cid*.

Summarizing the above, an *eth* is master if and only if (iff) all heartbeats received in the last *tSlave* time units are from *eth*s with a *cid* not lower than its own - the reception of a message with a lower *cid* value immediately brings the *eth* back to the slave role.

2.2 Assignment of Logical Addresses

To keep record of the *cid*s of all *lio*s, each *eth* maintains an array *cidTable* that is indexed with *cid*s from range [4...13] and contains *hwid*s as values. For a *cid* x from the above range, an *eth* then assumes that $cidtable[x]$ is the *hwid* of the *lio* to whom x is assigned as *cid*. If $cidtable[x] = -1$, then x is assumed to be unassigned.

The assignment of *cid*s is performed by the master *eth*, while slaves only update their *cidTable*s based on received messages. The statechart model of the *cid* assignment is showed in Fig. 2 for both masters and slaves. These models can be interpreted as refinement of the corresponding composite states (containing this way sub-machines) in the model of master election.

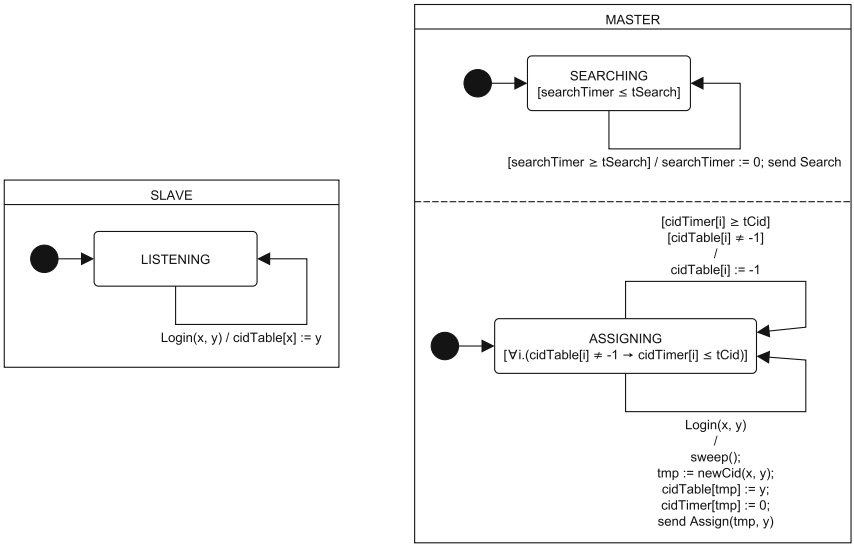


Fig. 2. Assignment of logical addresses as slave (a) and master (b)

The role of a slave *eth* is simply to keep track of assigned *cid* values by listening to *Assign* messages sent by the master and updating its *cidTable* based on them.

The behavior of a master *eth* can be summarized as follows (note that here *searchTimer* is a clock variable and *cidTimer* is an array of clock variables).

- Every *tSync* time units it broadcasts a message *Search*. As a response, each *lio* is supposed to send a message *Login*(x, y) where x is the current *cid* of the *lio* (-1 if undefined) and $y \neq -1$ is its *hwid*.
- Upon receiving a message *Login*(x, y), the following steps are performed.
  1. By calling a method *sweep*(), any occurrence of a given *hwid* other than the first is erased from *cidTable*. As turned out during verification, this method is required for resilient operation of the protocol.
  2. Based on the entries in *cidTable*, a new *cid* is calculated by the function *newCid* so that the following conditions are met.
     - If y appears in *cidTable* as a value at some index, then the index is returned as result. Since *sweep*() ensures that each *hwid* is unique in *cidTable*, the result is well defined.
     - Else if $x \neq -1$ and x is unassigned then it is returned as result.
     - Else the lowest unassigned *cid* is returned. The existence of such a *cid* is ensured by *sweep*().
  3. The *cidTable* is updated, the corresponding timer in *cidTimer* is reset and a message *Assign* is sent with the new *cid*.
- Other than that, if a row of *cidTable* corresponding to an assigned *cid* was not updated in the last *tCid* time units, then the *cid* gets unassigned.

2.3 LIOs

The model of a *lio* is shown on Fig. 3.

- Initially, the *lio* has no *cid* assigned ($cid = -1$).
- Upon receiving a message *Search*, the *lio* replies with a message *Login*(*cid*, *hwid*).
- Upon receiving a message *Assign*(x, y), if $hwid = y$, then *cid* is updated to x. The message is ignored otherwise.

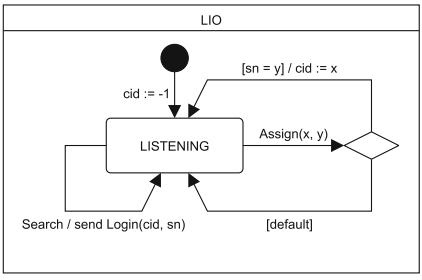


Fig. 3. Behavior of LIOs

3 Background

The formal background of our work is given in this section. The behavior of the system is formalized (on the basis of the SysML statecharts) as a transition system. The requirements of the protocol are described as formulas in linear temporal logic.

3.1 Transition Systems

Transition systems are widely used for the formal modeling of the behavior of reactive systems. A transition system over a set of atomic propositions AP is a tuple $TS = (S, T, I, L)$ such that

- S is a set of states,
- $T \subseteq S \times S$ is a transition relation,
- $I \subseteq S$ is the set of initial states and
- $L : S \longrightarrow 2^{AP}$ is a labeling function.

L can be extended to sequences of states in the obvious way. We assume that T is left total, that is for all $s_1 \in S$ there exists $s_2 \in S$ such that $(s_1, s_2) \in T$. Moreover, throughout this paper, we assume that each state $s \in S$ is an n-tuple of values of state variables $v_i \in V$ from the respective domain, and AP is a set of ground atomic predicates over V (i.e., these predicates do not contain free variables).

The behavior of a transition system can be characterized in terms of its initial traces. A path of TS is an infinite sequence of states $s_0 s_1 s_2 \ldots$ such that $(s_i, s_{i+1}) \in T$ for all $i \geq 0$. An initial path is a path where $s_0 \in I$. A trace induced by a path π is $L(\pi)$. An initial trace is a trace induced by an initial path.

3.2 Linear Temporal Logic

Linear temporal logic is widely used for the formal definition of the expected properties of a system.

A (propositional) linear temporal logic (LTL) over AP with temporal connective F (*finally*) has formulas of the form

$$\varphi ::= \mathsf{true} \,|\, a \,|\, \neg\varphi \,|\, \varphi \vee \varphi \,|\, \mathsf{F}\varphi$$

Here, $a \in AP$. Let $\sigma = A_0 A_1 A_2 \ldots$ be an infinite sequence such that $A_i \subseteq AP$ for all $i \geq 0$. Let the postfix $A_i A_{i+1} A_{i+2} \ldots$ be denoted by $\sigma[i \ldots]$. Then the satisfiability relation of the logic $\models$ is the smallest relation satisfying the following rules.

$$
\begin{array}{lll}
\sigma \models \mathsf{true} & & \\
\sigma \models a & \text{iff} & a \in A_0 \\
\sigma \models \neg\varphi & \text{iff} & \sigma \not\models \varphi \\
\sigma \models \varphi \vee \psi & \text{iff} & \sigma \models \varphi \text{ or } \sigma \models \psi \\
\sigma \models \mathsf{F}\varphi & \text{iff} & \sigma[i...] \models \varphi \text{ for some } i \geq 0
\end{array}
$$

Other usual Boolean connectives false, $\wedge$, $\rightarrow$ and $\leftrightarrow$ can be expressed in terms of $\neg$ and $\vee$. Temporal connective G (*globally*) is defined as $\mathsf{G}\varphi \triangleq \neg\mathsf{F}\neg\varphi$. It semantics is

$$
\sigma \models \mathsf{G}\varphi \quad \text{iff} \quad \sigma[i...] \models \varphi \text{ for all } i \geq 0
$$

A property expressed by a formula of the form $\mathsf{FG}\varphi$ where φ is a propositional formula is called a persistence property, and φ is called its persistence condition. According to the semantics of the temporal connectives, $\sigma \models \mathsf{FG}\varphi$ iff there exists some $i \geq 0$ such that for all $j \geq i$ we have $A_i \models \varphi$. Informally, a persistence property with condition φ expresses that the system eventually stabilizes to the configuration defined by φ.

Given a transition system TS and a temporal logic formula φ over AP, the model checking problem is to show that $\sigma \models \varphi$ for all initial traces σ of TS, denoted $TS \models \varphi$, or give a counterexample.

3.3 Abstraction

Abstraction is a key element in the efficient verification of complex systems as it reduces the state space by omitting irrelevant aspects of the system model. One such technique is *cone of influence reduction* (COI) [5] that proved its efficiency in several tools and approaches. It removes state variables that do not influence variables included in the temporal logic specification by building a dependency graph based on the transition relation. COI is a property preserving abstraction (hence the name reduction), that is, the system satisfies the property iff the reduced system does so.

4 Verification Approach

In this section we introduce the verification approach used for the analysis of the distributed protocol.

In general, the verification process of a fault tolerant system consists of many modeling and model checking steps. First, the system has to be verified leaving any fault assumptions out of consideration, thus the formal model of the fault-free system has to be developed. After the successful verification of the fault-free system, to verify fault tolerance, possible faults and their effects on the system have to be taken into account. Hence fault models have to be defined, that composed with the model of the fault-free system represent the behavior of the system under the given fault assumptions.

Since the verification of all possible faults and their combinations is often infeasible, at this point the verification engineer may restrict the range of investigated faults to selected ones. However, omitting any relevant fault or combination of faults can lead to verification results that cannot be justified with respect to the behavior of the real system.

In this section we introduce a different approach, which is based on the following assumptions and restrictions:

- Permanent and crash faults are not modeled since the focus is the verification of resilience, i.e., resuming the correct behavior of the system after transient faults. Permanent and crash faults are easier to detect than transient faults and need redundancy to provide fault tolerance.
- The effects of transient faults are modeled on a logical level as disturbances in the behavior of the related components in the form of additional transitions (called fault transitions) between states of the fault-free model. With regard to the common fault classification (crash, omission, timing, computation and Byzantine faults) we have the following considerations. Crash faults are not modeled as mentioned above. Omissions are covered by fault transitions that step over the omitted processing steps (including message sending or message processing). The effects of delayed messages and corrupt messages are covered by the combination of fault transitions that cause the loss of the original message and creation of a faulty one. Similarly, data corruption is covered by fault transitions that alter the state variables. Control flow errors among states, including the restart of the component, are also covered by fault transitions. Regarding Byzantine faults, those faults are covered whose effects can be modeled in terms of transitions between the states of the fault-free model.
- The resilience of the system is expressed as a persistence property: the effects caused by a transient fault shall be tolerated in such a way that after the occurrence of a fault (and the related disturbance), the behavior will eventually resume the correct one (this way almost all states along a path will belong to a correct behavior).

As presented in the following sections, the second restriction allows a systematic verification of faults, without requiring separate (manual) modeling of each fault. The third restriction enables in certain cases the use of a decomposition approach that divides the verification task into smaller and simpler ones.

In the following the used notations are introduced then the proof strategy for the efficient verification of fault models is detailed. Finally, the decomposition of persistence properties into simpler properties is given.

4.1 Notation

We introduce the following notations for two different restrictions of a transition system with respect to a propositional formula. Let $TS = (S, T, I, L)$. Then $TS_\varphi = (S, T, S|_\varphi, L)$ and $TS^\varphi = (S|_\varphi, T|_\varphi, S|_\varphi, L|_\varphi)$. Here, we define $S|_\varphi = \{s \in S \mid L(s) \models \varphi\}$, $T|_\varphi = T \cap (S|_\varphi \times S|_\varphi)$ and $L|_\varphi : S|_\varphi \longrightarrow 2^{AP}$, where

$L|_\varphi(s) = L(s)$. For example, $TS_{\mathsf{true}} = (S, T, S, L)$, that is, TS with all states considered as potential initial states.

It is easy to see that $(TS^\psi)_\varphi = (TS_\varphi)^\psi$, thus in this case the brackets can be omitted. Moreover, $(TS^\varphi)^\psi = TS^{\varphi \wedge \psi} = TS^{\psi \wedge \varphi} = (TS^\psi)^\varphi$ and $(TS_\varphi)_\psi = TS_\psi$.

4.2 Modeling Transient Faults

Let $TS = (S, T, I, L)$ be a transition system over AP. We model a fault in TS as a set of transitions $F \subseteq S \times S$ such that $F \cap T = \emptyset$, where a fault transition $(s, s') \in F$ models the effects of the occurrence of the fault in state s. In other words, we consider transient faults that can be expressed in terms of a nondeterministic change of state in the fault-free system. Naturally, the range of faults that can be modeled this way depends on the formulation of the system.

Given TS and F, we can define a transition system TS_F that models the system with a finite number of possible occurrences of transient fault(s) F as $TS_F = (S_F, T_F, I_F, L_F)$ over AP where

- $S_F = S \times \mathbb{N}$. Given a state (s, n), n is the number of transient faults that can still occur in the system.
- $I_F = I \times \mathbb{N}$. Initially, any finite number of faults are allowed to occur.
- $L_F((s, n)) = L(s)$.
- T_F is the smallest relation defined by the following rules (using a natural deduction style notation for antecedents and consequent):

$$\frac{(s, s') \in T \qquad n \in \mathbb{N}}{((s, n), (s', n)) \in T_F} \text{ normal transition}$$

$$\frac{(s, s') \in F \qquad n \in \mathbb{N}}{((s, n+1), (s', n)) \in T_F} \text{ fault transition}$$

To verify that a system TS satisfies a persistence property $\mathsf{FG}\varphi$ even if a transient fault defined by F can occur finitely many times, the following direct approach can be applied:

1. Construct TS_F from TS and F.
2. Check $TS_F \models \mathsf{FG}\varphi$.

However, the fact that the system TS_F satisfies a persistence property $\mathsf{FG}\varphi$ often originates from the stronger property that TS stabilizes to φ-states starting from *any of its states*. Using the above notation, this can be expressed by the following rule.

$$\frac{TS_{\mathsf{true}} \models \mathsf{FG}\varphi}{TS_F \models \mathsf{FG}\varphi} \text{ fault abstraction}$$

It is easy to see that this approach is sound, that is, if the antecedent hold, then the consequent also holds.

Proof. We prove the stronger property that $L_F(\pi) \models \mathsf{FG}\varphi$ for all paths of TS_F. Assume $TS_{\mathsf{true}} \models \mathsf{FG}\varphi$ and let $\pi = (s_0, n_0)(s_1, n_1)(s_2, n_2)\ldots$ be a path of TS_F. We apply induction on n_0. If $n_0 = 0$, then $L_F(\pi) = L(\pi)$ is an initial trace of TS_{true}, thus the statement holds. Now assume $n_0 > 0$. If $(s_i, s_{i+1}) \in T$ for all $i \geq 0$, the same applies as in the base case. So assume there is a state (s_{i-1}, n_{i-1}) with a minimal i such that $(s_{i-1}, s_i) \in F$. Since $n_i < n_{i-1}$, by the induction hypothesis, $L_F(\pi)[i\ldots] \models \mathsf{FG}\varphi$, thus $L_F(\pi) \models \mathsf{FG}\varphi$.

Since the rule is sound for any F, it allows the verification of fault tolerance without the need of explicitly modeling faults.

4.3 Decomposition of Persistence Properties

The resilience of the system is expressed as a persistence property $\mathsf{FG}\varphi$. The verification of such properties is a complex task as the model checker has to handle all traces and check if they contain fair cycles (with fairness constraint $\neg\varphi$) as counterexamples. In the following, we describe two rules that in definite cases enable the simplification of the model checking problem of such properties. We omit soundness proofs due to their simplicity.

The first rule describes the decomposition of a persistence property according to the expected behavior of the system. Without loss of generality, we can assume that the persistence condition is of the form $\varphi \wedge \psi$. Here, both φ and ψ define some configuration of the system that is expected to eventually persist. If the persistence of the system with respect to φ depends on its persistence with respect to ψ, the following rule can be applied to simplify the model checking problem.

$$\frac{TS \models \mathsf{FG}\varphi \qquad TS^{\varphi} \models \mathsf{FG}\psi}{TS \models \mathsf{FG}(\varphi \wedge \psi)}\ \mathsf{FG}\text{-detachment}$$

Here, all states of TS^{φ} are φ-states. The main advantage of such a decomposition is that if φ and ψ refer to different variables of the system, then the subproblems can be simplified significantly by abstractions that depend on the property, such as cone of influence reduction.

The second rule divides the model checking problem into two simpler problems.

$$\frac{TS \models \mathsf{F}\varphi \qquad TS_{\varphi} \models \mathsf{G}\varphi}{TS \models \mathsf{FG}\varphi}\ \mathsf{G}\text{-detachment}$$

Here, the check of $TS \models \mathsf{F}\varphi$ is a query searching for a lasso shaped initial path of $(\neg\varphi)$-states (as counterexample). The check $TS_{\varphi} \models \mathsf{G}\varphi$ basically amounts to verify whether φ is inductive, which is a less expensive step.

5 Verification of the Protocol

This section details the application of the approach presented in the previous section in the verification of the protocol. The formal, dense time model of the system was constructed as a network of timed automata [1,4], whose operational semantics can be expressed in terms of a transition system [4]. The verification aims at proving the resilience of the system: even in the presence of transient faults, the components shall be able to communicate with each other. This requires that after a finite number of faults, the system will persistently have a unique master and all *lio*s have a logical address assigned in a consistent way. Among others, this formulation admits the verification of correctness in the presence of the following transient faults:

- An *eth* or *lio* restarts.
- The content of an *eth*s *cidTable* changes.
- The *cid* of a *lio* changes.
- The content or recipient of a message changes.
- A message is lost.
- A message is created.

5.1 Decomposing the Verification of the Protocol

To enable model checking, the statechart model containing the composite statecharts of all *eth*s and *lio*s is mapped to a network of timed automata. Signal events are handled by an automaton representing a bounded capacity communication channel that is able to store and delay the sent messages until their reception. The resulting formal model can be analyzed by the model checker UPPAAL [3].

As the protocol has two functionalities (master election and assignment of communication IDs), the requirement of resilience is a composite property that includes the temporal correctness of these functionalities. Accordingly, resilience is formalized as a persistence property $\mathsf{FG}(\varphi \wedge \psi)$, where φ expresses that there is a unique master in the system, whereas ψ states that each *lio* was assigned a unique logical address that corresponds to a row of the masters *cidTable*.

The following proof tree shows the decomposition of this top level requirement.

$$\dfrac{\dfrac{TS_{\mathsf{true}} \models \mathsf{FG}\varphi \qquad TS^{\varphi}_{\mathsf{true}} \models \mathsf{FG}\psi}{TS_{\mathsf{true}} \models \mathsf{FG}(\varphi \wedge \psi)}\ \mathsf{FG}\text{-detachment}}{TS_F \models \mathsf{FG}(\varphi \wedge \psi)}\ \text{fault abstraction}$$

Instead of verifying the system model with different fault configurations, we employ the fault abstraction rule: this simulates that the verification starts after the occurrence of any finite number of transient faults. The next reduction rule splits up the property according to the FG-detachment rule: in the protocol, master election is a precondition for the successful logical address assignment. By proving the subproperties we can infer the validity of the property itself. Now, the task is to prove two properties referring to different aspects of the system.

– $TS_{\mathsf{true}} \models \mathsf{FG}\varphi$ expresses that the system initialized in any state will have a master and the participants will not change their role.
– $TS^{\varphi}_{\mathsf{true}} \models \mathsf{FG}\psi$ expresses that the system initialized in any state will finally have consistent *cid* assignment, assuming there is a unique stable master.

In the following sections the proofs of these two properties are detailed.

5.2 Verification of Master Election

The verification of the master election protocol is reduced to the model checking of the $\mathsf{FG}\varphi$ temporal logic specification on system TS_{true}. Now, the rule G-detachment can be applied, and thus the resulting model checking queries to be proven are $TS_{\mathsf{true}} \models \mathsf{F}\varphi$ and $TS_{\varphi} \models \mathsf{G}\varphi$.

As these resulting temporal logic formulas refer to only some aspects of the system, cone of influence abstraction can be employed to construct transition system TS_1 from TS_{true}. Behavior related to *cid* assignment is not relevant in the verification of master election: no interaction in master election is triggered or influenced by the administration of *cid* assignment. This enables the cone of influence reduction to fully reduce the model to the following elements, that are included in the model TS_1:

– Four *eth*s (with behavior as in Fig. 1).
– Communication channel.

The model $(TS_1)_{\varphi}$ is the same as TS_1, the only difference is that the initial states are those where the master has already been elected.

The property to be verified is φ, which refers to the situation of successful master election:

– eth_0 is master.
– eth_1, eth_2 and eth_3 are slave.

The formal proof tree that was applied in the verification of the master election protocol is the following:

$$\dfrac{\dfrac{TS_1 \models \mathsf{F}\varphi}{TS_{\mathsf{true}} \models \mathsf{F}\varphi} \quad \dfrac{(TS_1)_{\varphi} \models \mathsf{G}\varphi}{TS_{\varphi} \models \mathsf{G}\varphi}}{TS_{\mathsf{true}} \models \mathsf{FG}\varphi}\ \mathsf{G}\text{-detachment}$$

5.3 Verification of Logical Address Assignment

The verification of the logical address assignment protocol is reduced to the model checking of temporal logic specification $\mathsf{FG}\psi$ on system $TS^{\varphi}_{\mathsf{true}}$. Similar to the verification of the master election protocol, the rule G-detachment can be applied to decompose the problem into two parts. The resulting model checking queries to be proven are $TS^{\varphi}_{\mathsf{true}} \models \mathsf{F}\psi$ and $TS^{\varphi}_{\psi} \models \mathsf{G}\psi$.

In transition system $TS^{\varphi}_{\mathsf{true}}$, the master election procedure is assumed to have been successful, thus in the verification of *cid* assignment we can exploit that there will be no more changes in the roles of the *eth*s. In addition, the resulting temporal logic formulas refer only to aspects of the system related to *cid* assignment. These advantages of the decomposition can be exploited and cone of influence reduction can be applied to construct transition system TS_2 from $TS^{\varphi}_{\mathsf{true}}$, where TS_2 contains:

- eth_0 as master (with behavior as in Fig. 2).
- Ten *lio*s (Fig. 3).
- Communication channel.

The property to be verified is ψ, which refers to the situation where the *lio*s have unique *cid* values and it is consistent with the knowledge of the master:

- For each two rows of $eth_0.cidTable$, if they contain an equal value, then both values are -1 (thus the assigned *cid* values in the table of the master are unique).
- The *cid*s assigned to *lio*s correspond to the values in $eth_0.cidTable$.
- Each *lio* has a *cid* different from -1.

The formal proof tree that was applied in the verification of the *cid* assignment protocol is the following:

$$\dfrac{\dfrac{TS_2 \models \mathsf{F}\psi}{TS^{\varphi}_{\mathsf{true}} \models \mathsf{F}\psi} \qquad \dfrac{(TS_2)_{\psi} \models \mathsf{G}\psi}{TS^{\varphi}_{\psi} \models \mathsf{G}\psi}}{TS^{\varphi}_{\mathsf{true}} \models \mathsf{FG}\psi}\ \mathsf{G}\text{-detachment}$$

5.4 Result of the Verification

The verification problem was decomposed according to the proof rules detailed in the previous sections. COI abstraction was applied to the formal models, which significantly reduced the size of the formal models. When the first version of the protocol design was verified, insufficiencies were revealed in the protocol: an oscillation between states could occur that prevented the proof of the liveness property regarding the successful *cid* assignment. After the required modification of the design (among others the inclusion of the sweep() function) the UPPAAL model checker could then verify all the four tasks successfully within seconds. Without the proposed approach, namely the decomposition and abstraction steps, the verification could not succeed due to the resource limitations.

6 Related Work

Ensuring the correctness of safety critical systems is an important, but challenging task, which has an increasing number of successful attempts as the formal

verification tools and approaches become more and more mature. The verification of a startup protocol used in the same system as described in this paper was introduced in [11]. As an example of another safety critical real-time system with successful formal verification we mention a safety logic of a nuclear power plant [2]. In that approach time was discretized, which was not required in our current work. There were many attempts to verify the timed behavior of distributed protocols, for example, the real-time scheduling of a time triggered protocol was verified in [9]. In our paper, the verification was done by the UPPAAL model checker, which has many successful applications, see for example [6,10].

The approach introduced in this paper integrated many advanced techniques from the field of formal verification, among others abstraction, separation of aspects (decomposition) and cone of influence reduction.

Our fault modeling approach is similar to the one applied in the FSAP and xSAP framework [3], our contribution is the specific fault abstraction tailored to the persistence property.

Decomposition approaches proved their efficiency even for complex systems with a huge number of variables, for example in the verification of the latest Intel processors [8]. We applied similar ideas in our work but in the distributed systems domain. The verification of the FlexRay protocol is somewhat similar to our problem: in [7] authors developed an approach for abstracting data dependent and timed behavior of the system. In our approach we used the same data abstraction in combination with cone of influence reduction.

7 Conclusions

In our study we addressed the verification of a protocol in a distributed safety critical system. The protocol has to ensure communication between components of a SCADA system even in the presence of faults. In the model of the protocol, the great number of state variables, the time dependent behavior and the message-based asynchronous communication between components led to the well-known state space explosion problem, making the "brute force" verification intractable. To tackle these problems we devised an approach which combines the decomposition of the temporal specification with abstraction.

Fault abstraction is used to construct a single formal model that covers the effects of various transient faults that may disturb the operation of the protocol. This abstract model includes all behaviors of the system where a finite number of transient faults is allowed to occur. We proved the soundness of the approach.

We introduced two decomposition rules for persistence properties in linear temporal logic which are tailored to the problem domain. When applying these rules, we exploited the composite structure of the system functionalities (behaviour) to obtain simpler subtasks where the system could be simplified significantly by cone of influence reduction. By using the introduced approach, the verification of the protocol was successfully elaborated.

Acknowledgments. This work was partially supported by Gedeon Richter Plc. It was also supported by the ARTEMIS JU and the Hungarian Research and Technological Innovation Fund in the frame of the R5-COP project.

References

1. Alur, R., Dill, D.L.: A theory of timed automata. Theor. Comput. Sci. **126**(2), 183–235 (1994)
2. Bartha, T., Vörös, A., Jámbor, A., Darvas, D.: Verification of an industrial safety function using coloured petri nets and model checking. In: Proceedings of the 14th International Conference on Modern Information Technology in the Innovation Processes of the Industrial Entreprises (MITIP 2012), pp. 472–485. Hungarian Academy of Sciences, Computer and Automation Research Institute (MTA SZTAKI) (2012)
3. Behrmann, G., David, A., Larsen, K., Hakansson, J., Petterson, P., Yi, W., Hendriks, M.: Uppaal 4.0. In: Third International Conference on Quantitative Evaluation of Systems (QEST 2006), pp. 125–126 (2006)
4. Bengtsson, J.E., Yi, W.: Timed automata: semantics, algorithms and tools. In: Desel, J., Reisig, W., Rozenberg, G. (eds.) Lectures on Concurrency and Petri Nets. LNCS, vol. 3098, pp. 87–124. Springer, Heidelberg (2004)
5. Clarke Jr., E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press, Cambridge (1999)
6. D'Argenio, P., Katoen, J.P., Ruys, T., Tretmans, J.: The bounded retransmission protocol must be on time!. In: Brinksma, E. (ed.) TACAS 1997. LNCS, vol. 1217. Springer, Heidelberg (1997)
7. Gerke, M., Ehlers, R., Finkbeiner, B., Peter, H.-J.: Model checking the FlexRay physical layer protocol. In: Kowalewski, S., Roveri, M. (eds.) FMICS 2010. LNCS, vol. 6371, pp. 132–147. Springer, Heidelberg (2010)
8. Kaivola, R., Ghughal, R., Narasimhan, N., Telfer, A., Whittemore, J., Pandav, S., Slobodová, A., Taylor, C., Frolov, V., Reeber, E., Naik, A.: Replacing testing with formal verification in Intel CoreTM i7 processor execution engine validation. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 414–429. Springer, Heidelberg (2009)
9. Pike, L.: Modeling time-triggered protocols and verifying their real-time schedules. In: Formal Methods in Computer Aided Design (FMCAD 2007), pp. 231–238 (2007)
10. Ravn, A.P., Srba, J., Vighio, S.: Modelling and verification of web services business activity protocol. In: Abdulla, P.A., Leino, K.R.M. (eds.) TACAS 2011. LNCS, vol. 6605, pp. 357–371. Springer, Heidelberg (2011)
11. Tóth, T., Vörös, A., István, M.: K-induction based verification of real-time safety critical systems. New Results in Dependability and Computer Systems. AISC, vol. 224, pp. 469–478. Springer International Publishing, Switzerland (2013)

Considering Execution Environment Resilience: A White-Box Approach

Stefan Klikovits[1,2(✉)], David P.Y. Lawrence[1,3], Manuel Gonzalez-Berges[2], and Didier Buchs[1]

[1] Université de Genève, Centre Universitaire d'Informatique, Carouge, Switzerland
{stefan.klikovits,david.lawrence,didier.buchs}@unige.ch
[2] CERN, European Organization for Nuclear Research, Geneva, Switzerland
{stefan.klikovits,manuel.gonzalez}@cern.ch
[3] Honeywell International Sarl., Rolle, Switzerland

Abstract. Over the last decade code-based test case generation techniques such as combinatorial testing or dynamic symbolic execution have seen growing research popularity. Most algorithms and tool implementations are based on finding assignments for input parameter values in order to maximise the execution branch coverage. Only few of them consider dependencies from outside the Code Under Test's scope such as global variables, database values and subroutine calls as influences to the execution path. In order to fully test all possible scenarios these dependencies have to be taken into account for the test input generation. This paper introduces ITEC, a tool for automated test case generation to support execution environment resilience in large-scaled, complex systems. One of ITEC's corner stones is a technique called semi-purification, a source code transformation technique to overcome limitations of existing tools and to set up the required system state for software testing.

Keywords: Resilience · Automated test case generation · Software testing · Semi-purification · Execution environment resilience

1 Introduction

At the *Large Hadron Collider* (LHC), its experiments and several other installations at CERN physicists and engineers employ a *Supervisory Control And Data Acquisition* (SCADA) system to mediate between operators and controllers/front-end computers which connect to the sensors and actuators. As such applications require the configuration of hundreds of controllers, CERN has developed two frameworks on top of Siemens' *Simatic WinCC Open Architecture* (WinCC OA) [7] SCADA platform to facilitate their creation.

Due to lack of tool support for the WinCC OA's scripting language *Control* (CTRL) [8], it was so far not possible to write and execute unit tests in an efficient manner. Recently, the Industrial Controls Engineering group at CERN started the development of such a unit testing framework to fill this need. However, after more than ten years of development, CERN is left with over 500,000 lines

A. Fantechi and P. Patrizio (Eds.): SERENE 2015, LNCS 9274, pp. 46–61, 2015.
DOI: 10.1007/978-3-319-23129-7_4

of CTRL code for which only a very small set of unit tests exist. Hence, the verification of the source code remains a mainly manual task leading to high testing costs in terms of manpower and slower release times.

This situation is especially tedious during the frequent changes in the WinCC OA execution environment. Before every introduction of a new operating system version, the installation of patches or the release of a new framework version the code base needs to be re-tested. Over the lifetime of the LHC, these environment changes happen repeatedly (often annually) and involve a major testing overhead.

This paper introduces the *Iterative TEst Case* system (ITEC), a system for the automatic generation of test cases for the frameworks. ITEC relies on existing *automatic test case generation* (ATCG) methodologies such as dynamic symbolic execution [1,2] or combinatorial testing [3,4] to generate test input. Currently, many ATCG methodologies do not support the modification of source code dependencies, such as global variables, external resources (e.g. databases) and function/subroutine calls. While during manual unit test creation set-up routines and test doubles [11] (mocks, stubs, etc.) are frequently used to prepare and simulate a desired system state, techniques such as random testing and combinatorial testing and their respective tool implementations often do not support generation of test doubles and set-up routines. To overcome this caveat we introduce a technique called *semi-purification* which creates a bridge between existing testing tools and the mentioned requirements. Semi-purification is a process that replaces dependencies with additional function parameters, so that ATCG tools can generate test input for all code execution paths.

The rest of this paper is structured as follows: Sect. 2 describes the development style and applications at CERN and illustrates the resulting problems. Section 3 gives an overview of ITEC and describes the general work- and information flow. Section 4 explains semi-purification, in detail, followed by the introduction of some non-trivial semi-purification challenges in Sect. 5. Section 6 concludes.

2 Problem Description

The EN-ICE group at CERN has been developing and maintaining two frameworks for the development of SCADA applications for accelerators, experiments and infrastructure. WinCC OA, the underlying platform, provides amongst many other features the functionality to obtain and display data from sensors and send commands to actuators. Figure 1 depicts the typical three layers schema (field objects/sensors - frontend controllers - SCADA) of control systems at CERN and their connection to the operator work stations.

Currently about 600 individual WinCC OA systems are in action for the LHC and its experiments alone[1] – all of them relying on the functionality of the frameworks. Additionally many more applications use the frameworks, such

[1] Gonzalez-Berges, M. – Presentation at ETM Day, (CERN, 2013).

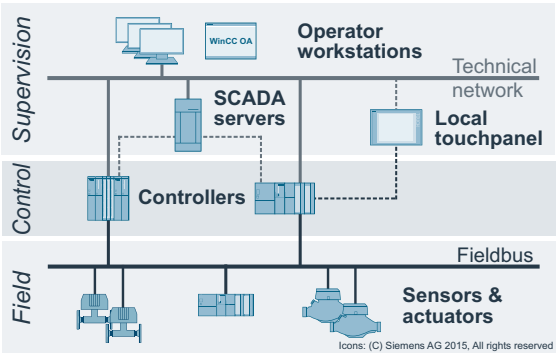


Fig. 1. Layer model dedepicting the connection of field objects, frontend controllers (e.g. PLCs) and Operator Work Stations (OWS) through SCADA applications

as the electrical network supervision with its tens of thousands devices and hundreds of thousands measurement and control points.

The frameworks' libraries and UI controls are primarily written in WinCC OA's proprietary scripting language *Control* (CTRL). CTRL's syntax is based on ANSI C, but provides some specific extensions such as for accessing the WinCC OA database. Other language modifications include the removal of features such as *pointers*, *structs* and *typedefs* and introduction of *reference parameters*, *dynamic arrays* and implicit type casting.

The syntax modifications are significant enough to exclude the use of existing unit testing frameworks. Hence, to date, mainly manual testing has been employed for verification. It has been considered to translate CTRL and use existing unit test case frameworks, but the backlog of not unit test covered code would still remain significant.

At the time of writing, CERN supports the execution of WinCC OA on two *operating systems* (OS) and typically up to two versions each (currently *Windows 7*, *Windows Server 2008 R2* and *Scientific Linux CERN 6*). As CERN's security protocol requires the switch to current operating system versions, regular upgrades are essential. The operators also introduce patches and updates in frequent intervals. Additionally, new WinCC OA releases are published every one to two years. Due to Siemens' support policy it is necessary to upgrade to new versions, which often involves updating the software because of changed features and/or compatibility issues.

This all results in an ever changing execution environment over the entire runtime of the LHC (over 30 years in total). Before every change, the software has to be tested and adapted if necessary. Due to the frameworks' sizes and the lack of automated CTRL test cases, this task takes a long time to complete.

The current release testing process requires developers to thoroughly verify the functionality of all components and fix any defects discovered. After several iterations of the test-fix cycle, the code base reaches a point that is deemed ready for shipment.

Although the manual verification has been complemented by a few automated user interface tests, testing remains a repetitive manual process, consuming time that developers could spend more productively on other tasks.

3 Iterative Test Case Generation System

To overcome the lack of test cases and code coverage, we propose the *Iterative TEst Case* (ITEC) generation system. ITEC generates and executes test input for pieces of source code, referred to as *Code Under Test* (CUT). The size of the CUT can vary in granularity from individual procedures to multiple functions connected by subroutine calls.

As displayed in Fig. 2, ITEC's workflow is split into four subtasks, each individually essential for the quality of the overall result.

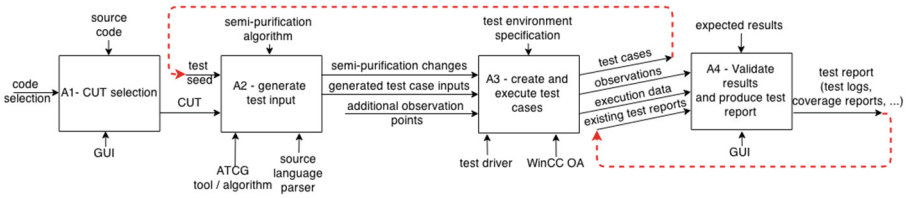


Fig. 2. Overview of the system's standard workflow on a high level with all in- and outputs of the system

A1. In the first step, the user chooses the *Code Under Test* (CUT). This can be a single function, a library or an entire component.

A2. Before the test case generation, verified test cases ("test seeds") can optionally be fed into the system to gather information from previous generations and existing (possibly manually implemented) test cases.

The test input generation itself is based on modified versions of the CUT. These modifications are made by a process we call *semi-purification*, which assures that external dependencies such as global variables, database values and other libraries are taken into account for the generation.

After the test data generation the resulting test input has to be cleansed from unviable values. The cleaning is necessary, as due to the transformation of the CUT it is possible that third party ATCG methodologies produce values that are impossible for the original code base.

A3. Based on the generated test input, the next task is to create and execute test cases. The required test setup procedures will be derived from the analysis of the semi-purification changes in *A2*. Subsequently, the test cases are executed and the changes to the system recorded. These observations will be added as assertions to the test cases.

A4. As the correctness of the assertions generated in *A3* has not been verified yet, ITEC will take advantage of existing test reports from previous test runs. These reports are used to assist in the decision to accept/refute the test observations (results). However, as there might not be enough test results available, or the existing information could be misleading, the final decision lies with the test engineer who serves as the test oracle.

After the judgment, the user has the choice to select a subset of the generated tests cases to be stored as future regression tests. Additionally a test report containing information about the generated test cases and the test execution logs is produced.

4 Generating Test Cases

The generation of test cases has experienced growing research interest. Although already introduced in the 1970s and 1980s the recent gain in computing power made the usage of methodologies such as adaptive random testing [6], combinatorial test case generation [3,4] and dynamic symbolic execution [1,2] more feasible. Leaning onto the insights of existing quality assurance research in the areas of specification based testing [12], test selection [13] and also quality assurance for resilient systems [14], we aim to build on these *Automatic Test Case Generation* (ATCG) processes for our purposes. In most cases ATCG methods produce a set of different test inputs that can be mapped on the input parameters of the function under test. The aim is to produce test inputs that execute the *Code Under Test* (CUT) along as many different execution paths as possible.

For this purpose we have to first define the notion of a function, for which we create test cases.

Definition 1 (Pure function). *We define a function* $f = \langle \pi, impl \rangle$ *to be a program function, that depends on its input parameter names* $\pi \in \Pi$ *and its implementation* $impl \in Impl$*, and consists of a sequence of statements describing its behaviour.*

Definition 2 (Input parameter value assignment). *Before the execution of a function* f*, it is necessary to assign values to its input parameters. This assignment is defined as follows:*

$$assign : \Pi \rightarrow V \tag{1}$$

Definition 3 (Function execution). *We define exec as the execution of a function* f *with a given inputs assignment in a system with a certain state. The result of this function execution is a deterministic return value.*

We define a unit test case for such a function in the following manner:

Definition 4 (Test case). *A test case* $t = \langle i, e \rangle$ *consists of* $i = assign(\pi)$*, an assignment of input parameter values, and* e*, the expected value for the return value.*

However, in many cases the CUT is not a pure function [5], meaning that next to the parameters Π, f has a set of additional dependencies D from outside its function scope, such as global variables or access to external resources through the file system or databases.

Definition 5 (Function dependencies). *The dependencies d of a non-pure function f is a set of variables and resources, defined outside f's scope part of the system state. ($d \subseteq state$). For our purposes we assume we can access the variable/resource value through $val(d_i); d_i \in d$.*

Further, the function can change the system state by modifying global variables and resources. These so-called observable side effects are defined by the set S, where S is the set of changes to the system produced by the execution of a function.

Definition 6 (Non-pure functions and side effects). *Let f be a non-pure function defined with parameters $\pi \in \Pi$, dependencies $d \in D$ and observable side effects $s \in S$. Then the execution of f with an input parameters assignment on a system with a given state, is the transition of the system to a new state and the production of a return value.*

$$exec(f, assign, state) \rightarrow r, state' \tag{2}$$

s is the set of new values that have been changed by the transition outside the function's scope.

$$s = state' \backslash state \tag{3}$$

For our purpose we will denote a non-pure functions as $f = \langle \pi, d, impl \rangle$.

We therefore extend our previous definitions to define test cases for non-pure functions.

Definition 7 (Test case for non-pure functions). *Given a non-pure function under test f, a test case $t = \langle p, i, e \rangle$ for f consists of the preparatory procedure p, the input value assignment i and the expected values e for some or all of the observations of the return value and the side-effects $r \cup s$.*

In order to keep the test cases of such a function deterministic, the system has to be brought into a known state before execution. That means that a preparatory procedure must be executed in order to set each dependency to the required value.

Definition 8 (Preparatory procedure). *The execution of a preparatory procedure $p \in Impl$ of a test case for a non-pure function with dependencies d modifies the state of the system so that each $d_i \in d$ returns the desired value.*

Many ATCG methodologies (e.g. random testing, combinatorial testing) perform their generations in a black-box fashion based on a function's signature but ignore that external dependencies in the function body influence its behaviour too. Therefore, to effectively cover all necessary combinations of parameters

and dependencies we have to modify the CUT using a technique called *semi-purification*. This ensures that the generation will take these dependencies into consideration. The generated values for the additional parameters can then be used to replace the dependencies with test doubles.

4.1 Removing Dependencies Through Semi-purification

Leaning onto the notion of pure functions, we refer as *semi-pure* to those procedures whose outcome only depends on its input values, but may have a certain number of side effects. Accordingly, semi-purification is the process of converting a not semi-pure function into a semi-pure one. During the semi-purification process dependencies from outside the CUT's scope are discovered and replaced by new input parameters. Since the resulting functions only depend on their input parameters, we can then employ standard ATCG methodologies to create test input.

To start with, we define the semi-purification process for external resources and global variables. These efforts are derived from the concept of *localization*. Localization is a refactoring technique that completely replaces the access to global variables and has been first picked up programmatically by Sward and Chamillard [9] for Ada programs. The topic was revisited for C programs by Sankaranarayanan and Kulkarni [10]. Leaning onto this technique, we not only replace global variables, but also database and other resource accesses with new input parameters.

Definition 9 (Semi-purification). *Semi-purification is the process of converting a function's dependencies d into additional parameters π.*

$$SP : D \rightarrow \Pi \tag{4}$$

For simplification, we can imagine that when SP is applied onto an entire function $f = \langle \pi, d, impl \rangle$, it converts some (or all) of the dependencies.

$$SP : \Pi \times D \times Impl \rightarrow \Pi \times D \times Impl \tag{5}$$

$$SP(\pi, d, impl) = \langle \pi \cup \Delta\pi, d', impl' \rangle \tag{6}$$

where $d' \subseteq d$ and $\Delta\pi = \{SP(d_i) | d_i \in d\}$.

In our case, we want to remove all global variables and external resources, hence after the semi-purification $d' = \emptyset$. We require this process to be neutral, meaning that the return values and side effects are the same when executing the functions.

Definition 10 (Semi-purification neutrality). *SP has to be implemented so that it is neutral for any given f. Neutrality is given, iff*

$$\forall assign, \forall \Delta assign, \forall state : exec(f, assign, state) = exec(SP(f), \{assign, \Delta assign\}, state') \tag{7}$$

where $state' \subseteq state$ and $\forall \Delta\pi_i \in \Delta\pi : val(d_i) = \Delta assign(\Delta\pi_i); \Delta\pi_i = SP(d_i); d_i \in d$.

It is to be said that it is possible to check this neutrality but it is hard to prove it. We rely that the experiences and assertions of our predecessors [9,10] are valid. At a future stage we will aim to investigate this matter deeper. Our tool will use an algorithm that will work according to the properties described in this paper, however, at the time of writing the implementation phase is not completed.

Example of Semi-purification. The following subsection shows a small example of the process of semi-purification. Listing 1 shows a non-pure function that relies not only on its input values, but also on the values that are obtained from a database (`dbGet(x)`) and a global variable (`GLOBAL_VAR`). To deterministically test `f(x)` it is necessary to execute the preparatory procedure before executing the function call to the CUT. In our example test case (Listing 2) we do this by explicitly setting the values of the global variable and the data point.

Listing 1. A non-pure function

```
1  f(x){
2    if GLOBAL_VAR:
3      return dbGet(x)
4    else:
5      return -1
6  }
```

Listing 2. A test case for a `f(x)`

```
1  test_f(){
2    dbSet("test",5) // prepare
3    GLOBAL_VAR = True
4    x = f("test") // act
5    assert(x == 5) // assert
6  }
```

To convert `f(x)` into a semi-purified function we introduce additional input parameters (`a` and `b`) for its dependencies. The references to `GLOBAL_VAR` and the database access (`dbGet(x)`) are replaced by `a` and `b`, respectively. The semi-purified version of the function from the previous example can be seen in Listing 3. The according test case (Listing 4) does not require a preparatory procedure.

Listing 3. Semi-purified `f(x)`

```
1  f_sp(x,a,b){
2    if a: //GLOBAL_VAR:
3      return b // dbGet(x)
4    else:
5      return -1
6  }
```

Listing 4. Test case for `f_sp(x,a,b)`

```
1  test_f_sp(){
2    x = f("test",True,5) // act
3    assert(x == 5) // assert
4  }
```

Recursive Application of Semi-purification. Oftentimes the CUT is not an individual procedure, but includes subroutine calls to perform its operation. In these cases the semi-purification has to be applied recursively and the changes made to a subroutine (*callee*) need to be propagated to the *caller* in order to keep the CUT valid.

Listing 5. Recursive Semi-purification

```
1  functionA(x){
2    a = functionB(x)
3    return a
4  }
5
6  functionB(x){
7    b =  GLOBAL_VAR
8    b++
9    return b
10 }
```

Listing 6. Semi-purified CUT

```
1  functionA(x,y){
2    a = functionB(x,y)
3    return a
4  }
5
6  functionB(x,y){
7    b = y
8    b++
9    return b
10 }
```

An example of this behaviour can be seen in Listing 5 which displays the CUT (`functionA`). The semi-purification of the subroutine (removing the dependency to `GLOBAL_VAR`) modifies the function signature of `functionB`. This change has to be incorporated into the function call in `functionA` and also added to `functionA`'s signature. The resulting code can be observed in Listing 6.

4.2 Creating Test Inputs and Preparatory Routine

Following the semi-purification process we can execute ATCG tools on the modified CUT. The result is a set of test case inputs that could be used to verify the semi-purified version of the CUT. However, we aim to generate regression tests for the original CUT and hence need to re-convert the output.

The result of applying ATCG methodologies onto semi-purified functions, is a power set of assignments of values to the input parameters $assign' : (\pi \cup \Delta\pi) \rightarrow V$. For simplicity, instead of one assignment $assign'$, we will imagine the elements of the output as two separate mappings $assign$ and $\Delta assign$, one for the original parameters and one for the additional semi-purified parameters.

Definition 11 (Test input generation for semi-purified functions). *We define the test input generation TIG for a semi-purified function f' that produces a power set of parameter value assignments, $assign : \Pi \rightarrow V$ and $\Delta assign : \Delta\Pi \rightarrow V$.*

$$TIG : f \rightarrow \mathcal{P}(assign) \tag{8}$$

$$TIG(\pi \cup \Delta\pi, \emptyset, impl) = \bigcup_{i \in 0,\ldots,n} \{assign_i, \Delta assign_i\} \tag{9}$$

However, as the target is to produce test cases for the original CUT, the generated test inputs need to be re-transformed accordingly. This means that we have to take the value assignments for the newly introduced parameters in f' and convert them into a preparatory procedure.

Definition 12 (Inverse Semi-purification). *Given a function f and its corresponding semi-purified version f' we define SP^{-1} as the operation that converts $\Delta assign$ into a preparatory procedure P.*

$$SP^{-1} : \Delta assign \rightarrow P \tag{10}$$

where the execution of a $p \in P$ asserts that each $d_i \in d$ returns the value defined in $\Delta assign(SP(d_i))$.

The preparatory procedure includes for example the setting of global variables, data point values or the preparation of other external resources such as the file system state. Figure 3 shows the schematic representation of the information flow for the test input generation.

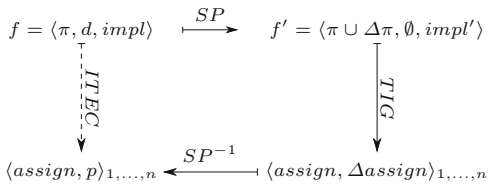


Fig. 3. Using ATCG on non-pure functions by removing dependencies on global variables and external resources.

4.3 Semi-purification of Subroutine Calls

Using the above described method, it is possible to create functions that only depend on their input parameters. Recursive application of semi-purification on subroutines permits the generation of test input for function interactions. However, when testing programs with a deep control flow graph, the semi-purification process might easily result in a procedure with dozens of input parameters and many subroutines that have to be rewritten. Whereas the approach we introduced so far permits the generation of integration tests, this approach comes with caveats. The generation of test inputs for CUTs of this complexity is computationally expensive, as for most ATCG methodologies the complexity rises with the number of input parameters. Additionally, in many cases the function calls connect multiple libraries, voiding the initial goal of creating unit tests.

To overcome this limitation, we introduce a semi-purification process that removes a function's *subroutine calls* (SRC). The two processes can be seen as complementary actions, each removing different dependencies. For the rest of this paper we continue with the following notation: Global variable and resource dependencies remain D, subroutine dependencies are identified by D_{SRC}. SP_{SRC} is the process of replacing some of these subroutine calls with additional input parameters $\Delta\Pi'$. The set of remaining subroutines D'_{SRC} contains SRCs that should not be replaced (e.g. built-in string operations or hashing procedures). For certain CUTs and situations this set is empty.

Definition 13 (Semi-purification of Subroutine Calls). *Semi-purification SP_{SRC} of subroutine calls is the process of converting a function under test (with subroutine calls) $f = \langle \pi, d \cup d_{SRC}, impl \rangle$ into a function $f'' = \langle \pi \cup \Delta\pi', d \cup d'_{SRC}, impl' \rangle$ where $\Delta\pi'$ is the set of newly introduced parameters to replace some (or all) of the subroutine dependencies d_{SRC}.*

$$SP_{SRC} : \Pi \times D \times Impl \rightarrow \Pi \times D \times Impl \tag{11}$$

$$SP_{SRC}(\pi, d \cup d_{SRC}, impl) = \langle \pi \cup \Delta\pi', d \cup d'_{SRC}, impl' \rangle \tag{12}$$

where $d'_{SRC} \subseteq d_{SRC}$ and $\Delta\pi' = \{SP(d_i) | d_i \in d_{SRC}\}$.

ATCG methodologies can then generate the input for the function with fewer dependencies (or none at all). The resulting input assignments

$(assign, \Delta assign')$ have to be re-transformed into test input for the original function. The additional test input for replaced subroutine calls ($\Delta assign'$) needs to be converted into a preparatory procedure P' that create test doubles for the masked out SRCs.

The entire workflow combining the two semi-purification routines (SP and SP_{SRC}) is displayed in Fig. 4.

Definition 14 (SP_{SRC} neutrality). *SP has to be implemented so that it is neutral for any given f. Neutrality is given, iff*

$$\forall assign, \forall \Delta assign, \forall state : exec(f, assign, state) = exec(SP(f), \{assign, \Delta assign'\}, state') \tag{13}$$

where $state' \subseteq state$ and

$$\forall \Delta\pi'_i \in \Delta\pi' : val(d_i) = \Delta assign'(\Delta\pi'_i); \Delta\pi'_i = SP(d_i); d_i \in d \tag{14}$$

Definition 15 (SP^{-1}_{SRC}). *The inverse semi-purification of SRCs is defined as the generation of a preparatory procedure $p' \in Impl$ for Δd_{SRC}.*

$$SP^{-1}_{SRC} : \Delta assign' \rightarrow P' \tag{15}$$

so that the execution of a $p \in P$ asserts that each subroutine $d_i \in d_{SRC}$ returns the value defined in $\Delta assign'(SP(d_i))$.

For subroutines the creation of a preparatory procedure includes the specification of test doubles for these functions that provide the specified values.

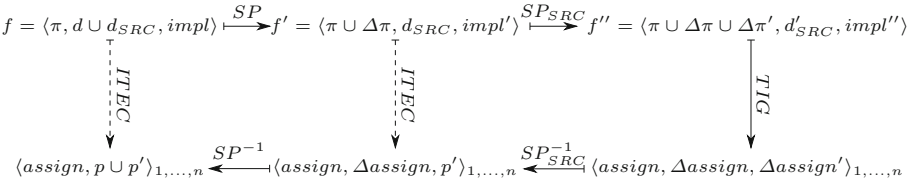


Fig. 4. The full semi-purification workflow, including subroutine replacement

5 Problematic Areas of Semi-purification

The application of semi-purification reaches limits in certain areas. The following section will introduce some of these areas and show our plans to overcome these problems.

5.1 Loops

One such situation is the presence of subroutine calls inside iterations (loops and recursions).

Looking at the semi-purified function `sleepUntilReady` in Listing 7 where the parameter `a` has been newly introduced to replace the dependency on `dbGet(notReadyDP)`.

Listing 7. Loop example

```
1  sleepUntilReady(a){
2    while a : // replaces dbGet(notReadyDP)
3      sleep(5) // sleep for 5 seconds
4  }
```

The algorithm that we introduced so far is not sufficient to fully test the functionality of this function. ATCG methodologies would create two test inputs (`True` and `False`) for `a`. However, the case where the function is executed with `a = True`, the test case would result in an endless loop.

In order to generate a test case for the `a = True` we have to change the parameter `a` to be a list of values and the loop to access a new element of this list as shown on the Listing 8. Using these modifications it is possible to generate the test input to achieve the desired behaviour.

Listing 8. Modified loop

```
1  sleepUntilReady(a){
2    i = 0
3    while a[i] :
4      sleep(5)
5      i++
6  }
```

Please note, that the example above can lead to an *IndexOutOfBounds* error, that we have to account for. In general it is not possible to create (single thread) test cases for unbounded loops, so we have to transform the loop to be bounded. ATCG tools work around the limitation of unbound loops by having certain timeout constraints on the time or number of execution paths analysed. We aim to investigate this issue in the future by instrumenting the loops and drawing conclusions from this execution information.

5.2 Dependencies Between Subroutines

Another problematic scenario is in the case of dependencies between subroutines. Listing 9 shows such a CUT, containing two read and one write access to a global variable (`SPEED_VAR`). The semi-purification algorithm we introduced above would create the CUT presented in Listing 10.

The problem with the naïve approach of semi-purification is that each access to a dependency is replaced with a new input parameter. Here `a` and `b` access the same variable which's value cannot change in standard (single thread) executions. For this reason the semi-purification process should replace them with the same parameter. The write access remains unmodified as it is a side effect and there are no further reads from this dependency.

Listing 9. CUT with dependencies

```
1  adjustSpeed(){
2    x = getTheSpeed()
3    if x < 10 :
4      doubleTheSpeed()
5  }
6
7  getTheSpeed(){
8    return SPEED_VAR
9  }
10
11 doubleTheSpeed(){
12   speed = SPEED_VAR
13   SPEED_VAR = speed*2
14 }
```

Listing 10. Naïvely semi-purified CUT

```
1  adjustSpeed(a,b){
2    x = getTheSpeed(a)
3    if x < 10 :
4      doubleTheSpeed(b)
5  }
6
7  getTheSpeed(a){
8    return a // SPEED_VAR
9  }
10
11 doubleTheSpeed(b){
12   speed = b  // SPEED_VAR
13   SPEED_VAR = speed*2
14 }
```

Additionally, it is in general necessary to discover sequential write-read scenarios, where an external value is first set, then read. In this cases it is necessary to re-use the value from the write for the read as well, as otherwise the behaviour of the modified CUT is changed.

5.3 Concurrency

The example from the previous subsection shows the generation of impossible scenarios for single-thread execution. However, WinCC OA uses so-called "managers" to perform tasks. Each manager has its own context and works as an individual process.

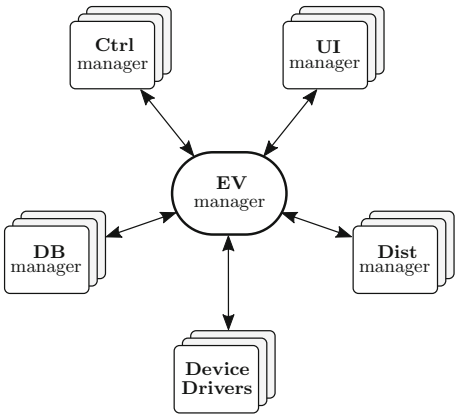


Fig. 5. WinCC OA's manager concept

Figure 5 displays some of the WinCC OA managers such as the *event manager* (EV), the *database manager* (DB) and the *Control manager* (Ctrl). The EV serves as message router and keeps an in-memory image of the current database values, handles alarms and executes functions on the data points.

Each manager is directly connected to the EV and hence it is possible that two processes modify the same resource without noticing the other one's updates.

Figure 6 schematically displays such a behaviour where `Process1` sets a data point value while `Process2` modified the state meanwhile. This scenario becomes inherently more complex, considering that CERN's control systems consist of hundreds of subsystems, which each resemble the one in Fig. 6 and can access each other's data points via communication through the *Dist managers*. To prevent this, it is possible to ask for a lock on a data point and stop other managers from modifying it.

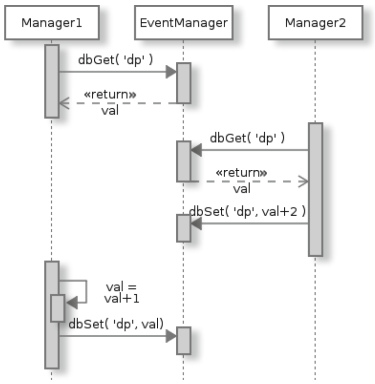


Fig. 6. Process1 overwrites the data point without noticing the changes

For the semi-purification it is essential to detect these situations and avoid race conditions and dirty reads and writes, but also identify when locks are being used.

ITEC has to be capable of generating test cases for these situations, meaning that while single thread unit tests are essential, ITEC should also allow for generating test cases to prevent concurrency issues and race conditions to happen.

As most of these error-situations can only be observed through exceptions and error codes, it is necessary that the test doubles fulfil those needs too.

6 Conclusion

CERN's recent efforts to create a unit testing framework for the proprietary CTRL language opened the door for a modern quality assurance process. However, after over a decade of development, the backlog of source code without automated test coverage makes changes in the execution environment challenging.

To address this problem and increase the testing process resilience, we outlined ITEC, a testing system that has the purpose to automatically generate test cases at a unit level for existing source code. To that end, ITEC bases its efforts on existing *automated test case generation* techniques such as adaptive random testing, combinatorial testing and dynamic symbolic execution.

Unfortunately, a large majority of the techniques previously mentioned are not well suited for practical tests generation. As a matter of fact, dependencies on global variables, external resources and subroutine calls are usually disregarded. Therefore, to overcome these obstacles often encountered in CERN's source codes, we thoroughly and formally presented a novel technique called *semi-purification*, its goals being to strip the *code under test* (CUT) from dependencies that lie outside the considered scope. For that purpose, we first addressed the semi purification of code in the presence of global variables and external resources. We then quickly extended the semi-purification's scope to also take into account subroutine calls as dependencies.

Finally, we succinctly addressed additional problems that often hinder the use of ATCG techniques. Among these, we discussed the pertinent idea of considering system concurrency with unit tests and illustrated the needs to deal with these kind of situations by examining an existing problem over the considered system.

References

1. Qu, X., Robinson, B.: A case study of concolic testing tools and their limitations. In: 2011 International Symposium on Empirical Software Engineering and Measurement, pp. 117–126. IEEE Computer Society, Los Alamitos (2011)
2. King, J.C.: A new approach to program testing. ACM SIGPLAN Not. **10**(6), 228–233 (1975). ACM, New York
3. Nie, C., Leung, H.: A survey of combinatorial testing. ACM Comput. Surv. **43**(2), 11:1–11:29 (2011). ACM, New York
4. Colbourn, C.J.: Combinatorial aspects of covering arrays. In: Le Matematiche, vol. 58, Catania, Italy (2004)
5. Haskell. Functional programming (2014)
6. Anand, S., Burke, E.K., Chen, T.Y., Clark, J., Cohen, M.B., Grieskamp, W., Harman, M., Harrold, M.J., Mcminn, P.: An orchestrated survey of methodologies for automated software test case generation. J. Syst. Softw. **86**(8), 1978–2001 (2013)
7. ETM Professional Control: WinCC OA at a glance. Technical report, Siemens AG (2012)
8. ETM Professional Control: Control script language (2015). http://etm.at/index_e.asp?id=2&sb1=54&sb2=118&sb3=&sname=&sid=&seite_id=118. Accessed 18 Apr 2015
9. Sward, R.E., Chamillard, A.T.: Re-engineering global variables in Ada. In: Proceedings of the 2004 ACM SIGAda International Conference on Ada, pp. 29–34. ACM, New York (2003)
10. Sankaranarayanan, H., Kulkarni, P.: Source-to-source refactoring and elimination of global variables in C programs. J. Softw. Eng. Appl. **6**(5), 264–273 (2013)
11. Meszaros, G.: Test double patterns (Chapter 23). In: XUnit Test Patterns: Refactoring Test Code, pp. 521–590. Prentice Hall PTR, Upper Saddle River (2006)
12. Barbey, S., Buchs, D., Péraire, C.: A theory of specification-based testing for object-oriented software. In: Hlawiczka, A., Simoncini, L., Silva, J.G.S. (eds.) EDCC 1996. LNCS, vol. 1150, pp. 303–320. Springer, Heidelberg (1996)

13. Péraire, C., Barbey, S., Buchs, D.: Test selection for object-oriented software based on formal specifications. In: Gries, D., de Roever, W.-P. (eds.) PROCOMET 1998. LNCS (IFIP), pp. 385–403. Springer, New York (1998)
14. Lawrence, D., Buchs, D., Wellig, A.: Using instrumentation for quality assessment of resilient software in embedded systems. In: Majzik, I., Vieira, M. (eds.) SERENE 2014. LNCS, vol. 8785, pp. 139–153. Springer, Heidelberg (2014)

Engineering Cross-Layer Fault Tolerance in Many-Core Systems

Rem Gensh[1(✉)], Alexander Romanovsky[1], and Alex Yakovlev[2]

[1] Centre for Software Reliability, Newcastle University, Newcastle upon Tyne, UK
{r.gensh,alexander.romanovsky}@newcastle.ac.uk
[2] School of Electrical and Electronic Engineering, Newcastle University, Newcastle upon Tyne, UK
alex.yakovlev@newcastle.ac.uk

Abstract. Engineering modern many-core systems is a challenging task because of their scale and complexity. We cannot focus on ensuring their dependability without understanding its interplay with performance and energy consumption. This calls for developing new structuring mechanisms that step away from the traditional ways systems are developed (such as strict layering, strong encapsulation, abstractions, hiding). The paper reports on the initial steps of a PhD work focusing on development methods and tools for architecting cross-layer fault tolerance in many-core systems in which error detection and error recovery are applied at several system layers in a concerted coordinated fashion to ensure the overall system efficiency.

Keywords: Error detection · Error recovery · Performance · Power consumption · Abstractions · Encapsulation

1 Introduction

Fault tolerance [1] is the means of dependability, allowing us to prevent system failures in the presence of faults. To achieve this after an error is detected in a component of a computer system (e.g. hardware, operating system or software), an error recovery mechanism returns the system to the full or reduced system functionality.

Term cross-layer interaction [2] was introduced to refer to the idea that it is better suited to ensure system efficiency with respect to various non-functional characteristics by reasoning about system layers together, rather than by completely abstracting the functionality of individual layers and trying to improve the efficiency of each of them in isolation. The examples of such characteristics are resource usage, reliability, performance and power consumption.

Many-core systems are likely to become the predominant type of the architectures used in the future. According to [3] the number and variety of cores will be continually increasing. One of the challenges in the area is that there is a need to understand the trade-off between reliability and energy-consumption, as more energy is necessary to support the operation of redundant cores. Another challenge is that fault tolerance is typically developed on one system layer even though these systems are always built as multilayer architectures.

A. Fantechi and P. Patrizio (Eds.): SERENE 2015, LNCS 9274, pp. 62–70, 2015.
DOI: 10.1007/978-3-319-23129-7_5

Developing a useful and general support to help in engineering cross-layer fault tolerance is a challenge. The main problem is that in this work one needs to develop techniques that assist in breaking the conventional way the abstractions are created. This paper reports on the initial work in the PhD study conducted by the first author in the area of developing methods and tools to support engineering of cross-layer fault tolerance for the many-core systems.

2 TCP/IP as a Motivating Example

The TCP/IP stack (see Fig. 1) provides an excellent example of cross-layer fault tolerance applied to ensure fault tolerance and improved performance. All layers of the TCP/IP stack participate in error detection and recovery in a concerted fashion.

The Link layer, the lowest layer of the TCP/IP stack, is used to transmit the packets between the Internet layer interfaces of two nodes inside a local network segment. The Transport layer provides host-to-host communication for the Application layer. The Application layer supports data exchange between processes on different hosts over the network connection supported by the lower layers.

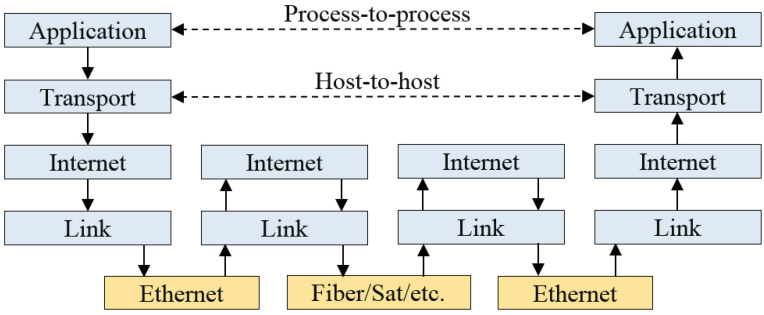


Fig. 1. TCP/IP stack

TCP provides reliable packets transmission, even though the packets may be lost, corrupted or delivered out-of-order. At the Link layer, the Ethernet frame contains a CRC-32 checksum: a received frame with an incorrect checksum is discarded. The main protocol at the Internet layer is IP (Internet Protocol), which has two implementations, IPv4 and IPv6. The header of the IPv4 packet is protected by CRC-16 checksum. The IP packets with wrong checksums are dropped by the receiver. The IPv6 header does not contain a checksum, assuming that Link layer provides an adequate error detection. The UDP and TCP packets of the Transport Layer have CRC-16 checksums, which protect the payload and addressing information.

TCP sends Acknowledgement to the sender to confirm the correct receipt or Negative Acknowledgement if the packet checksum is incorrect. In the latter case, the Automatic Repeat reQuest (ARQ) method is used to retransmit the corrupted packet. If the sender receives neither Acknowledgement nor Negative acknowledgement by timeout, it resends the packet. Such situation can happen when the packet is lost or

rejected by the lower layers due to incorrect checksum. In addition, a TCP packet contains a sequence number, which allows the receiver to discard duplicate packets and sequence reordered packets. This, in particular, shows that the errors of the lower layers are detected and recovered by concerted efforts at several layers.

At the Application layer, the developer can choose an appropriate Transport layer: either the connection oriented and reliable TCP or the connectionless UDP. The developer's choice between reliable data delivery and data delivery in time depends on the application requirements. If UDP was chosen, than it might be necessary to implement error detection and error recovery at the application layer by adding redundant data e.g. status code or encryption.

To conclude, TCP/IP is a useful example of how fault tolerance can be applied in coordination at several system layers. This was done to ensure its efficiency and flexibility. In our opinion the main factor contributing to the success of this protocol is the way the fault tolerance was designed and engineered.

3 Many-Core Systems

Computer systems with tens, hundreds or thousands processor cores are called many-core systems [4], whereas multi-core systems have typically only 2–8 cores. It is expected that these systems will replace multi-core systems in the near future and that they will become widely used in the safety-critical applications. Many-core architecture uses low performance small cores each of which alone is less productive than a large core, however hundred or thousand of small cores deliver better performance than ten large cores. Even though we can expect that the throughput will increase with the increasing number of cores, the performance growth is restricted by the percentage of serial code in the application (Amdahl's Law). Engineering of the efficient many-core systems is now an area of active research focusing on developing scalable methods for structuring complex many-core applications and for efficient parallelization at the OS and hardware layers.

Another challenge in developing these systems is ensuring their fault tolerance. The first problem is that when voltage and frequency scaling is applied to reduce power consumption the reliability is affected when near-threshold values are used. So we need to understand the interplay between energy and reliability. Moreover the modern semiconductors are more vulnerable to faults or negative effects like ageing and variation due to their extra small sizes. Many-core systems can provide redundancy to deal with these problems (e.g. some cores can be used to provide error detection and error recovery for other cores). Our analysis shows that in many-core systems fault tolerance is typically applied at the individual layers such as OS, application, communication middleware, memory, etc.

Ensuring high performance, low energy consumption, efficient resource utilisation and high reliability, as well as understanding their interplays are the main challenges for all types of many-core systems ranging from the large-scale systems, like data centres to the small-scale systems, like mobile devices.

4 Layered Fault Tolerance

Computer-based systems are prone to faults at different layers of the system stack, starting from circuitry degradation at the hardware layer to the bugs in the application source code. In designing large systems substantial efforts are being made to mitigate the effects of errors caused by faults at all layers of the system stack. Traditionally, the errors are handled at the layers where they are detected. Such an approach, reducing the complexity of system engineering, is very convenient for the developers and for the teams of developers as it simplifies system composition, reuse, maintenance and modification. This situation illustrates the predominance of convenience over the system efficiency in run time.

Let us look first into several examples of how fault tolerance of system components is typically ensured. Triple modular redundancy is a form of N-modular redundancy when three components perform the same operation and a single output is produced by a majority-voting system. The recovery block [5] works with several implementations of the same algorithm: after executing the primary variant, an acceptance test verifies the results. If the acceptance test fails, the system is rolled back and the secondary variant is tried. Eventually, either a variant passes the test or an exception handler is invoked. The N-version programming [6] is an approach aiming to reduce the probability of software faults by developing two or more functionally equivalent program versions independently in accordance with the same initial specification. These versions are executed concurrently and a special voting algorithm choses the correct output.

The two typical approaches used to ensuring fault tolerance at several layers are action nesting and extending component interfaces with exceptions. The best examples of the former are exception handling and nested ACID (Atomicity, Consistency, Isolation, Durability) transactions. The latter are best represented by F. Cristian's approach to providing recovery for modular software [7] and the idealised fault tolerance component pattern [8]. Even though these techniques support layered system structuring for fault tolerance they do not support concerted cross-layer fault tolerance at multiple layers when the decision to apply error detection and recovery is made for all layers together.

The substantial disadvantage of the layered approach is that the system layers are considered separately. Under such circumstances, it is impossible to adjust the layers in order to achieve optimal system operation in terms of performance, energy efficiency or resource utilization. Unnecessary error corrections are possible when the upper layer cannot specify the required quality of service of the bottom layer.

For example, let us consider a many-core system where the fault rate of one core is significantly larger than the rate of another core. When an error is detected, the error recovery could be achieved by re-executing the calculations on the same core making it slower. If there were a special cross-layer mechanism, which can make a decision that under some fault rate value, hardware layer error recovery should be applied, but after exceeding this value, it is necessary to inform OS about the faulty core, than the system fault tolerance as a whole would be more efficient. In the latter case, OS would be capable to hide the faulty core from the applications for some time.

The optimality and the effectiveness of the system fault tolerance could be achieved only when all the layers of the system are considered together. This is unfeasible when the strict layered approach is used.

5 Cross-Layer Fault Tolerance

The cross-layer fault tolerance assumes that fault tolerance mechanisms are distributed among all layers of the system stack and designed together (see Fig. 2). The final decision is made according to the whole system state rather than to the states of the individual layers separately.

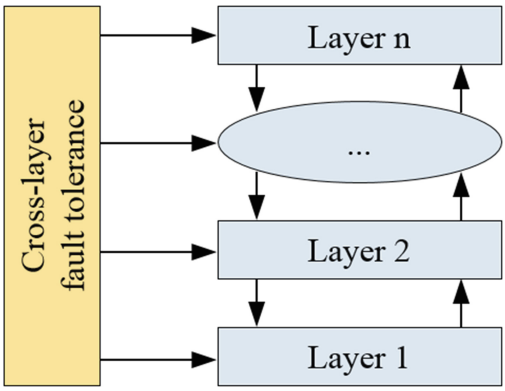


Fig. 2. Cross-layer fault tolerance

The Cross-Layer Reliability Visioning Study [2] proposes that it is necessary to use a cross-layer, full-system-design approach to reliability. The authors argue that in a cross-layer reliable system the entire system stack needs to collaborate in order to recover the errors and tolerate variations. This will be achieved because the relevant information about the system state is shared across the layers. In addition, the application domain of the system should always be taken into account, since different domains have various reliability requirements.

Study [2] introduces the cross-layer approach to the reliable system design, forecasting that the electronics industry is about to approach two inflection points that require drastic changes in integrated circuits design. The first point is reliability and predictability. In the fabrication technologies less than 65 nm gate leakage became a serious problem that led to reliability deterioration. This will push the designers to alter the assumptions that semiconductors and other microelectronic elements will operate without fails during the whole system lifetime. The second point is energy consumption, which is a crucial issue for contemporary computer systems. Paper [9] states that nowadays the entire Information and Communication Technologies sector consumes about 10 % of the energy generated in the whole world.

As mentioned in Sect. 2, the TCP/IP stack illustrates the practical usage of cross-layer approach to ensure system fault tolerance. The cross-layer design is now widely used in the area of the wireless sensor networks (WSNs). Since reliability, performance and energy consumption are crucial factors for these systems, the optimal operation of the whole system can only be guaranteed when the layers are considered together. Single layer approach cannot share important information among different layers. Consequently, each layer does not have complete information and it is impossible to achieve the optimal system operation. In addition, the single layer approach is incapable of adapting to the environmental change. Paper [10] discusses cross-layer adaptivity techniques, which leverage functionalities at different layers of the protocol stack. The application layer is frequently involved in these activities, supporting current system operation in accordance with measurements and forecasts of the monitored system. Study [11] proposes a new routing protocol based on the cross-layer principle in order to manage faults in wireless sensor networks, decrease signalling overhead and power consumption. A cross-layer data delivery protocol for delay/fault-tolerant mobile sensor networks was developed in [12]. The protocol aims to optimize energy consumption in the light of throughput requirement, stable connectivity of the sensor nodes and sufficient channel bandwidth.

The on-going work on cross-layer fault tolerance is patchy and is mainly focusing on the area of WSNs. Cross-layer fault tolerance is not applied in many-core systems, which will be the predominant architecture in the future. Unfortunately, development of cross-layer fault tolerance complicates the system design and it breaks the abstractions and needs a holistic approach. To make it practical the developers need to be assisted by novel system and software engineering techniques. The aim of this PhD study is to develop such techniques (architectures, models, patterns, libraries, tools) and to demonstrate that applying the cross-layer fault tolerance for many-core systems can improve performance and energy-efficiency.

6 Ongoing and Future Work

Several topics are being investigated during the first year of the study to understand better the domain and to develop an initial understanding of the requirements for cross-layer fault tolerance engineering in many-core systems.

6.1 Experiments with Odroid-XU3 Board

The Odroid-XU3 board is a small Octa-Core computing device implemented on energy-efficient hardware, which is based on the ARM big.LITTLE heterogeneous architecture and consists of a high performance Cortext-A15 quad core processor block (big), a low power Cortex-A7 quad core block (little), GPU and DRAM. The following experiments were carried out to understand the correlation between power consumption and performance. To clarify the voltage-frequency dependencies for the A7 and A15 power domains, the first experiment measured voltage, current and power at different frequencies without any additional workload. The second experiment involved

measuring the same parameters under 100 % load created by a stress test program executing 50 million square root operations, and brought unexpected results, that at the identical frequency, A7 was a bit faster than A15 and consumed four times less power. The same trend was observed with sine and cosine functions. Experiments with other operations gave the anticipated results when A15 was more than twice faster than A7 at the same frequency and almost three times faster at a maximum frequency. In the third experiment, we investigated the influence of thread sleep state (between active state periods) on energy consumption. Our results show that in terms of energy, it is more efficient to execute the task as fast as possible. Fourth experiment was held to investigate possible power and energy savings after disabling CPU cores for the cases when the workload is not very high. It was found that power consumption reduces more than 8 times after disabling all four cores of A15 processor. This technique can be used to reduce power and energy consumption of many-core systems during their idle time.

6.2 Threads Scheduling

In order to understand the behaviour of the scheduler at the ultimate load in Windows and Ubuntu OSs the threads scheduling experiments were carried out. Two or more threads with 10 ms tasks, requiring as much CPU time as possible were bound to one CPU core using thread affinity. It was observed that Windows and Ubuntu schedulers have different logic. Windows scheduler tries to run one thread to completion and after that switches to another thread, whereas Ubuntu scheduler tries to switch between different threads during execution. These findings should be taken into account while designing cross-layer fault tolerance for high performance and reliable many-core systems.

6.3 Global Exception Catch Block

Breaking the abstractions will lead to the situation when the encapsulation principle is violated. This, in turn, can be the reason for the inconsistent state. Let us consider the hypothetical global catch block, which will specify that all exceptions down the call stack should be propagated to this global catch block, even though there are catch blocks below that can handle these exceptions. For example, we have module 1, that calls function `Do` of module 2 inside the try-global-catch statement. An implementation of the `Do` function already has a standard try-catch statement inside. If an exception is thrown in function `Do` and the standard catch block in the same function has only error loggers and does not attempt any recovery or rollback, than module 2 remains consistent after the exception is propagated to the global catch block of module 1. However, if function `Do` has a transaction that should be rolled back in the catch block, than the state of module 2 could become inconsistent after exception propagation to the global catch block in module 1, since the catch block in the `Do` function will not be applied. This simple example illustrates that it is necessary to study the effects of error propagation through the system layers in order to understand the problems to be faced during development of cross-layer fault tolerance. We are now developing an advanced

exception handling scheme to support global exception handling as a programming mechanism for cross-layer fault tolerance.

6.4 Relax Framework

The authors of [13] propose a co-called language-level Relax mechanism for providing energy-efficient reliability and cross-layer fault tolerance for supercomputers. It allows the developer to specify code regions where low-reliability computations should be tolerated. In case of error, recovery is performed by re-execution of the "relaxed" code block. We plan to apply the similar approach on Odroid-XU3 board, by using little cores for detection of the calculation errors of big cores. If the error is detected, the recovery will be done by simple re-execution of the calculation. This technique will be useful for developing more sophisticated cross-layer fault tolerance mechanisms for many-core systems.

6.5 Future Work

Our short-term plans include work in the two areas. Firstly, we will apply the Order Graphs – the scalable approach developed in our group [14] - to model fault tolerance, power consumption and performance of many-core systems and to represent cross-layer fault tolerance. In particular, we would like to apply the idea of model fidelity to the area of fault/failure significance and of developing the corresponding cross-layer fault tolerance. Secondly, a medium-scale case study will be implemented to gain the experience in developing cross-layer fault tolerance for many-core systems.

Acknowledgments. This work is supported by the EPSRC/UK PRiME project and by the School of Computing Science, Newcastle University (UK).

References

1. Avizienis, A., Laprie, J.C., Randell, B., Landwehr, C.E.: Basic concepts and taxonomy of dependable and secure computing. IEEE Trans. Dependable Sec. Comput. **1**(1), 11–33 (2004)
2. DeHon, A., Carter, N., Quinn, H.: Final Report for CCC Cross-Layer Reliability Visioning Study. http://relxlayer.org/ (2011)
3. Borkar, S.: Thousand core chips—a technology perspective. In: Proceedings of the 44th Annual Design Automation Conference (DAC) (2007)
4. Vajda, A.: Programming Many-Core Chips. Springer, New York (2011)
5. Randell, B., Xu, J.: The evolution of the recovery block concept. In: Software Fault Tolerance. John Wiley & Sons Ltd, Hoboken, pp. 1–22 (1994)
6. Chen, L., Avizienis, A.: N-version programming: A fault tolerance approach to reliability of software operation. In: Proceedings of the Twenty-Fifth International Symposium on Fault-Tolerant Computing, pp. 113–119 (1995)

7. Cristian, F.: A recovery mechanism for modular software. In: Proceeding of the 4th International Conference on Software Engineering, ICSE'1979 (1979)
8. Anderson, T., Lee, P.A.: Fault Tolerance, Principles and Practice. Prentice/Hall International, New Jersey (1981)
9. Mills, M.P.: The Cloud Begins With Coal. CEO Digital Power Group, Washington D.C (2013)
10. Carnevali, L., Ridi, L., Vicario, E.: Stochastic fault trees for cross-layer power management of WSN monitoring systems. In: Proceedings of IEEE Conference on Emerging Technologies & Factory Automation, pp. 1–8 (2009)
11. Rachelin Sujae, P., Vigneshpandi, M.: A cross layer fault tolerant communication architecture for wireless sensor networks. Middle-East J. Sci. Res. pp. 1292–1296 (2014)
12. Wang, Y., Wu, H., Lin, F., Tzeng, N.F.: Cross-layer protocol design and optimization for delay/fault-tolerant mobile sensor networks (DFT-MSN's). IEEE J. Sel. Areas Commun. **26** (5), 809–819 (2008)
13. Ho, C.H., de Kruijf, M., Sankaralingam, K., Rountree, B., Schulz, M., de Supinski, B.R.: Mechanisms and evaluation of cross-layer fault-tolerance for supercomputing. In: Proceedings of the 41st International Conference on Parallel Processing (ICPP), pp. 510–519 (2012)
14. Rafiev, A., Xia, F., Iliasov, A., Gensh, R., Aalsaud, A., Romanovsky, A., Yakovlev, A.: Order graphs and cross-layer parametric significance-driven modelling. In: Proceedings of ACSD 2015. IEEE CS, Brussels (2015)

Risk Assessment Based Cloudification

Szilárd Bozóki[1(✉)], Gábor Koronka[2], and András Pataricza[1]

[1] Budapest University of Technology and Economics, Budapest, Hungary
{bozoki, pataric}@mit.bme.hu
[2] Eötvös Loránd University, Budapest, Hungary
kgabor@cs.elte.hu

Abstract. Design for resiliency always needs a proper trade-off between dependability of a system, and (cost) overhead. Cloud computing offers surplus resources at a favorable cost, thus (modular) redundancy based solutions became affordable for a broad spectrum of applications. The paper aims at a risk model based assessment of the benefit of applying redundant cloud resources.

Keywords: Resiliency · Cloud · Risk assessment

1 Introduction

Faults, errors and failures are omnipresent in human made systems. The huge risk potential originating in failures and outages in mission critical and hard SLA (*Service Level Agreement*) bound applications is traditionally well confined by means of architectural fault tolerance based, for instance, on modular redundancy. Similarly, business-critical services could suffer of losses up to millions of dollars per downtime hour [1], which makes rough granular redundancy, despite of its high cost, to a financially feasible optimum.

The joint characteristics of critical applications is the high value of the assets endangered by faults and their consequences, which drive the trade-off cost point high. On the contrary, the availability of cheap computing power provided by cloud computing pushes the cost balance point by orders of magnitude down, thus making redundancy based resilience to a favorite candidate for a broad spectrum of applications.

The typically low level of SLA guaranteed by the providers of public clouds is still a deal breaker barrier for professional applications, despite all the attractiveness of cheap cloud resources. Even private clouds cannot reach such an assured SLA like dedicated HA servers, thus the designers are afraid to build a cathedral on the sand by adopting cloud based solutions.

For instance, telecommunications solution providers, traditionally confined by the requirements of carrier grade services, are clearly on the brink of embracing the benefits of cloud computing through Network Function Virtualization (NFV) [2], in the case if a proper dependability, security and resilience can be guaranteed.

A. Fantechi and P. Patrizio (Eds.): SERENE 2015, LNCS 9274, pp. 71–81, 2015.
DOI: 10.1007/978-3-319-23129-7_6

Our motivating example, NFV is based on three core ideas: moving from proprietary telco specific boxes to standard hardware, decoupling the telco application environment from the HW platform by virtualization, and leveraging common datacenter and cloud technologies, in order to focus onto the telco specific parts. NFV is envisioned to offer groundbreaking benefits, such as reduced equipment and energy costs by consolidation and economies of scale, faster time to market by software implemented functionality reusing COTS components, and runtime resource optimization.

The logic interface (system boundary) hides the majority of details of the underlying platform from the cloud user. However, the SLA provided by a complex framework would be highly sensitive to platform failures unless an appropriate redundancy scheme is masking the insufficiencies of the underlying infrastructure. Moreover, in reality, cloud providers lack strict SLAs assuring well-defined and measureable KPIs. Even the relevant NFV documents are at an early stage [3].

The core question arising from this dichotomy: ***How can cloud services of varying quality be trusted by critical applications of high dependability requirements?***

This paper focuses on critical applications running on *virtual machines* hosted on *public clouds.* We interpreted the definition of dependability [4] ("*the ability of a system to avoid service failures that are more frequent or more severe than is acceptable."*) with a risk based approach, effectively transforming dependability requirement violations into a risk issue. Here the severity and rate of failures are quantified as the expected value of loss (risk) due to potential service outages. Either the failure rate or the severity needs to be confined in order to mitigate risk.

2 Cloud Computing Basics

The most widely accepted cloud computing definition is as follows [5]:

> *"Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction."*

The different, partly independent aspects exposed by the definition offer several opportunities for resiliency:

- *On-demand self-service* and *rapid elasticity* facilitate error *compensation* by diagnostics based on demand redundancy. Moreover, rapid re-provisioning supports *software (and platform) rejuvenation*.
- *Isolation* and *reconfiguration* based resiliency by exchanging suspicious resource instances to fresh ones selected from an independent part of the system. Note that reconfiguration may rely on the fast rearrangement of Software Defined Networks (SDN) instead of using the time consuming migration mechanisms. This promises the inclusion of soft real time systems into the application scope of clouds.

2.1 Service Models of Cloud Computing

Cloud applications depend on several subsystems. The definition of the system boundary between the cloud user and provider domains differs in the individual service models according to the level of abstraction of the platform services.

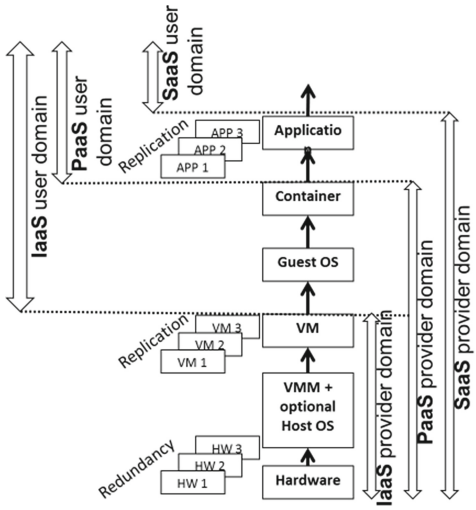


Fig. 1. Cloud service models and redundancy architectural pattern alternatives

Generally, the key layers are (Fig. 1): the infrastructure (consisting of the HW platform, host OS equipped with a virtual machine monitor –VMM), virtual platform (virtual resources, machines), guest OS, application container and the application itself. There are three corresponding primary service models of cloud computing: *Infrastructure as a Service* (IaaS), *Platform as a Service* (PaaS), and *Software as a Service* (SaaS).

The individual service models offer further means of fault tolerance, in addition to the previously mentioned ones, differing primarily in their respective granularity of fault management and resilience. For instance, IaaS supports *rollback* and *rollforward* between different VM images, *reinitialization* from an image, and *diagnosis*. PaaS may provide similar means, even supporting design per implementation diversity. SaaS typically elevates the services provided by the application framework (middleware).

The applicability of different resiliency mechanisms differs to a great extent by the ownership of the layers [5]. Self-operating a cloud (*Private cloud*) or part of it (*Hybrid cloud*) naturally grants observability and controllability of the provider domain, enabling more efficient *error detection,* both *concurrent* and *preemptive*. In contrast to this, cloud platforms operated by an independent organization (*Public cloud*) or a group of organizations (*Community cloud*) do not necessarily grant this, making them inferior from error detection point of view.

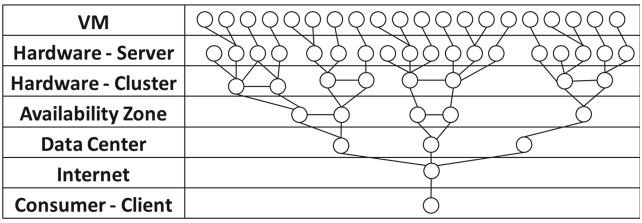


Fig. 2. Physical cloud model

The physical structure of cloud is also a major factor to be considered (Fig. 2). Cloud resources are mostly hosted in multiple locations (*regions*), which are further subdivided into physically isolated locations, named *availability zones* (AV zones), providing geographic diversity [6]. Low latency dedicated links connect the AV zones, whilst regions are interconnected via Internet, implying generally worse, less reliable and unsecure communication.

3 Risk Model

3.1 Objective and Scope

One potential answer to the fundamental question formulated in the introduction as: ***"How can cloud services of varying quality be trusted by critical applications of high dependability requirements?"*** is the exploitation of the affordable resources in the cloud in the form of modular redundancy (Redundancy architectural pattern) [7].

Narrowing it down, the actual question driving this paper is the following: **In order to optimize risk, how many redundant virtual machines are needed*?*** For simplicity, the concepts are illustrated for a simple fault tolerance scheme.

Our approach in answering the questions is based on three essential model elements: *cloud pricing*, used to quantify the exact cost of cloud resources, such as the operational cost of a VM (VM price/hour); the *cloud fault model*, used for downtime estimation; and the *cost of downtime* losses used for risk assessment.

3.2 Cloud Model

The target topology of interdependence follows the layers described above [8–11, 18]. Since regions and AV zones are physically separated, a tree structure is used there in contrast to the connectivity within an availability zone (Fig. 3).

A similar top-down structured interdependence graph as in [9] is used for basic fault impact analysis. Leading from the consumer (client) through regions (data centers) and availability zones (AV zone) to virtual machines (VM).

All nodes, except the consumer node have two states: *1 (up), 0 (down).* Nodes may fail and be repaired according to their respective time between failures τ_k and time to

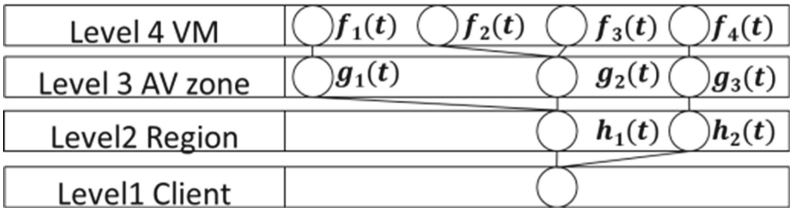


Fig. 3. Abstract cloud model

repair η_k stochastic distributions. We reuse distribution types and their actual parametrization from literature processing observation data [12].

Note that the characteristic distributions used here differ essentially from those used in traditional reliability analysis as no initial or aging faults appear in professional clouds during operation (thus Weibull styled approximations are unfaithful). Similarly, software dominated faults are long tailed and do not possess the forever young property making the exponential approximation unfaithful as well. Best fitting is achieved by applying resource type dependent Gamma distribution, widely used for life expectancy testing or if the times between Poisson distributed events are relevant.

Similarly, lognormal distribution describes best the complex multifactor repair process with heavy differences between hardware and software faults.

A complete but consistent set of failure related data is simply unavailable for the wide public. Accordingly, infrastructure related empirical data will be reused from publications observing a private data center while data related to regions originate in published data gained by external observers of public cloud services [13].

The basic principles, implementation and operational procedures are highly similar in different clouds independently of they are private or public nature. We still assume that a fundamental consistency between this data related to different cloud objects remains valid.

Note that the layered approach to fault tolerance patterns basically decomposes the calculations into two loosely coupled parts corresponding to the different layers.

We will reuse for VMs and the basic infrastructure the numerical parameters extracted of large number of observations [12] in the following pilot calculations: for instance, the Gamma distributed random variable denoting the time between failures τ_k corresponds to 19.6 h of MTBF, while lognormal random variable η_k has 37.22 days of MTTR. Overall, this results in a 97.85 % VM availability and failure rate of r_{VM} = 2.15 %.

Let the random variable $f_n(t)$ denote the state of the n^{th} node at time t (Fig. 4); If node n is an initially fault free VM (checked during provisioning) i.e. $f_n(0) = 1$, then:

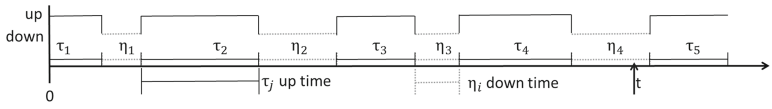


Fig. 4. Timeline and state change

If exactly f failure-repair actions are performed until the time instance t:

$$f_n(t) = 0 \quad \text{if } \sum_{k=1}^{f} \tau_k + \sum_{k=1}^{f-1} \eta_k \le t < \sum_{k=1}^{f} \tau_k + \sum_{k=1}^{f-1} \eta_k + \eta_f$$

$$f_n(t) = 1 \quad \text{if } \sum_{k=1}^{f-1} \tau_k + \sum_{k=1}^{f-1} \eta_k \le t < \sum_{k=1}^{f-1} \tau_k + \tau_f + \sum_{k=1}^{f-1} \eta_k \text{ where } f > 1$$

An application loses its underlying infrastructure if all the candidate VMs (or VM ensemble) needed by it are down or unreachable. This way the reliability block diagram (and its related fault tree in Fig. 5) is simply the serial connection of the region, the AV zone and the respective set of VMs (with potential redundant parallel blocks).

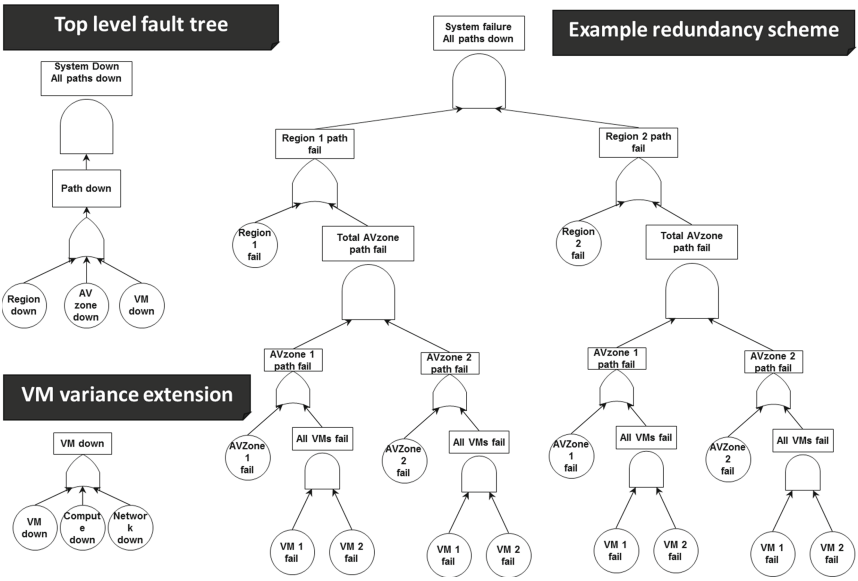


Fig. 5. Example fault trees

The top level fault tree (Fig. 5) is refined for the case of an example scenario with 2 regions, each composed of 2 AV zones, and two VMs in each AV zone. A system fails if all region paths fail. A region path fails if the region fails or all the zone paths fail. A zone path fails if the zone fails or all the VMs fail. Note that autonomous fault management performed by the built-in supervisory control of the cloud is for the sake of simplicity omitted of the model.

Let $g_n(t)$ and $h_n(t)$ denote random variables corresponding to the states of AV zones and regions at time t. The same sources were used for parametrization of the availability models of the AV zones as for the VMs for the sake of consistency [12]. This identified power and network outages as primary origins of down times in AV zones. The failure rate r_{AV} of AV zones (0.24 %) is 11 % of that of VMs.

The region failure rate (r_{RE}) is extracted from a one year long historic data on downtime events [13]. The examined data set had three primary attributes: provider name, region name, and downtime.

It is assumed that we have N regions N_i AV zones in the ith region. For simplicity, regions are assumed to be identical both in the size and failure characteristics. Let denote by N_{ij} the number of VMs deployed in the jth AV zone of the ith region.

The total system failure rate r_{total} becomes:

$$r_{total} = \prod_{i=1}^{N}(1-(1-\prod_{j=1}^{N_i}(1-(1-r_{VM}^{N_{ij}})(1-r_{AV})))(1-r_{RE})) \quad (1)$$

3.3 Downtime and Risk Assessment

Risk assessment using the fault model was based on different redundancy schemes differing in the number of VMs and their allocation (Table 1). Three significantly different cloud providers were selected using T-test with 0.025 significance level in order to avoid the overfitting of the results to a particular provider [13]. VM cost was based on the offering of a cloud provider [14]. The penalty of application downtime was assumed to be linear with the length of the outage [1, 15]. (for a more sophisticated value assessment of cloud see [16]) The expected annual total cost was calculated using the hourly cost of the VMs and the hourly cost of downtime.

3.4 Extending the Basic Model

The basic model was extended by the variance of network and compute resources [17]. This restriction would allow assessment of soft real time systems with timing constraints. A resource was considered down if its performance was below a threshold, respectively network was down over a latency limit, whilst compute was down when the computation time exceeded a given percentage of the mean computation time. The availability of resources is plotted against different thresholds in Fig. 6.

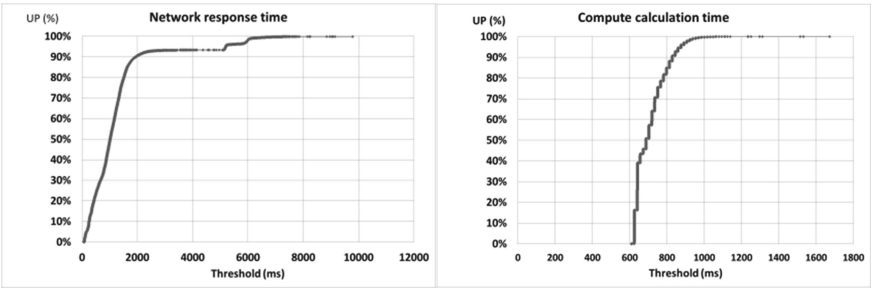


Fig. 6. Varying network and compute availability based on different thresholds

3.5 Calculations

The basic assumption is that the user has a control over:

- **Redundancy Degree**: the number of virtual machines used in modular replication based his fault tolerance scheme compensating independent faults in the individual VMs.

Table 1. Assessment of the basic and extended model

Failure rates	Redundancy schemes			Basic model		Extended with variance	
	#VMs	VMcost	Allocation	#9s	Expected total cost	#9s	Expected total cost
Provider A	1	157.68	1	1	37 687 436.18	1	47 399 861.36
Normal	2	315.36	1;1	3	811 006.55	3	1 282 696.59
VM 0.021470511	3	473.04	1;1;1	5	17 920.58	4	35 176.14
AV 0.002361756	4	473.04	2;2	6	1 072.84	6	1 672.52
RE 4.13813E-05	4	473.04	2;1	6	1 046.73	6	1 628.36
	4	473.04	1;1;1;1	6	1 032.13	6	1 595.64
Extended	5	832.20	1;1;1;1;1;	8	840.27	7	857.59
VM 0.027014362	5	814.68	2;1;1;1	8	823.44	7	841.43
AV 0.00297158	5	797.16	2;2;1	8	806.67	7	825.35
RE 4.13813E-05	5	788.40	2;3	7	833.53	7	870.94
	6	1007.40	1;1;1;1;1;1	10	1 007.57	9	1 008.09
Provider B	1	157.68	1	1	37 899 777.08	1	47 610 999.24
Normal	2	315.36	1;1	3	820 167.62	3	1 294 146.54
VM 0.021470511	3	473.04	1;1;1	4	18 217.01	4	35 641.84
AV 0.002361756	4	473.04	2;2	6	1 317.61	6	2 033.79
RE 0.00016524	4	473.04	2;1	6	1 152.83	6	1 797.32
	4	473.04	1;1;1;1	6	1 040.65	6	1 612.48
Extended	5	832.20	1;1;1;1;1;	8	840.50	7	858.17
VM 0.027014362	5	814.68	2;1;1;1	8	825.78	7	846.14
AV 0.00297158	5	797.16	2;2;1	8	812.02	7	835.29
RE 0.00016524	5	788.40	2;3	6	980.53	6	1 078.38
	6	1007.40	1;1;1;1;1;1	9	1 007.58	9	1 008.11
Provider C	1	157.68	1	1	38 305 773.99	1	48 014 695.97
Normal	2	315.36	1;1	3	837 826.91	3	1 316 180.59
VM 0.021470511	3	473.04	1;1;1	4	18 793.10	4	36 543.82
AV 0.002361756	4	473.04	2;2	6	1 935.12	5	2 873.95
RE 0.000402058	4	473.04	2;1	6	1 362.13	6	2 128.43
	4	473.04	1;1;1;1	6	1 057.36	6	1 645.30
Extended	5	832.20	1;1;1;1;1;	8	840.95	7	859.28
VM 0.027014362	5	814.68	2;1;1;1	8	830.48	7	855.49
AV 0.00297158	5	797.16	2;2;1	7	825.68	7	858.64
RE 0.000402058	5	788.40	2;3	6	1 411.16	6	1 624.53
	6	1007.40	1;1;1;1;1;1	9	1 007.59	9	1 008.14

- **Deployment for Diversity:** by populating copies into different cloud regions to avoid geographically correlated faults.

The calculations were based on the r_{total} formula (Eq. 1). The different numbers of VMs allocated to regions are delimited by semicolon. Penalty was $2*10^5$\$/ downtime-hour. As different providers apply similar technologies (with potentially different implementations) for the management of VMs and AV zones, we approximated their failure rates uniformly based on [12].

4 Related Work

[19] deals with different strategies from the provider view to optimally meet SLA requirements. Our work presented addresses a similar question, however from the perspective of a cloud tenant wanting to meet only his own SLA. While providers are capable and motivated to search for a global portfolio optimum, tenants of the cloud are usually incapable or disincentivized in it. Moreover, as previously mentioned, a provider has a wider variety of candidate means to use: underlying infrastructure, tenant SLA budget, implementation specific (vSpeher) fault and error management mechanisms etc. On the other hand, tenants could operate on the infrastructure of several providers, allowing to optimize the portfolio of offered resources with varying SLAs. This makes our work naturally different, but complementary.

[20] presents a detailed model of a queueing and scheduler based job execution service deployed on cloud infrastructure, which distributes jobs and their related tasks to worker loads, the dynamically provisioned virtual machines. In general, our aims and perspective are similar, the assessment of running applications (cloud user perspective) on a scalable cloud infrastructure. Compared, their work is more specialized and focused on the widely used queueing and scheduler based job execution services, while our present work is more general, and lacks application specific details.

[21] Presents an interesting analysis of cloudification based on the case study of deploying a scientific grid on a private cloud. It clearly indicates that the profitability of deploying a scientific grid architecture on top of a private cloud is a viable alternative if the environmental parameters (VM size, PM size, job characteristics etc.) are right. Although their aim of assessing cloudification is similar to ours, our cloud user on a public cloud based perspective is different from their private cloud case study.

5 Conclusion

The basic model presented considers only availability, showing that the cloud is ready for critical applications from a pure availability point of view. However, the extended model highlights the Achilles heel of current public cloud computing platforms, resource quality variance, especially for network resources.

Redundancy seemed a viable risk mitigation strategy capable of reducing annual total costs. The affordability of multiple modular redundant platforms pushes the boundary of criticality from the cloud to demanding commercial applications.

The model presented in the paper was slightly simplified for an easier understanding. It is able to support risk assessment, providing an estimate for the required redundancy depending on the value of the continuity of services.

Modern cloud technology offers several new opportunities for the implementation of a redundant architecture. Software defined networks designated for a fast reconfiguration are favorite candidates to substitute the heavily time consuming failover implementations based on migration.

Applications of changing criticality during their execution result in different levels of required redundancy. Dynamic redundancy allocation can provide a tradeoff between dependability and resiliency versus cost.

References

1. Kembel, R.: Fibre Channel: A Comprehensive Introduction. Northwest Learning Assoc., Tucson (2000)
2. Network functions virtualisation – introductory white paper. https://portal.etsi.org/nfv/nfv_white_paper.pdf (2015). Accessed 15 May 2015
3. Network functions virtualisation – service quality metrics V1.1.1. http://www.etsi.org/deliver/etsi_gs/NFV-INF/001_099/010/01.01.01_60/gs_NFV-INF010v010101p.pdf (2015). Accessed 15 May 2015
4. Avizienis, A., Laprie, J.-C., Randell, B., Landwehr, C.: Basic concepts and taxonomy of dependable and secure computing. IEEE Trans. Depend. Secur. Comput. **1**(1), 11–33 (2004)
5. Mell, P., Grance, T.: The NIST definition of cloud computing (2011)
6. AWS guide. http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-regions-availability-zones.html (2015). Accessed 15 May 2015
7. Hanmer, R.: Patterns for Fault Tolerant Software. Wiley Publishing, Chichester (2007)
8. Billinton, R., Allan, R.N.: Reliability Evaluation of Engineering Systems - Concepts and Techniques. Springer, New York (1992)
9. Jhawar, R., Piuri, V.: Fault Tolerance and Resilience in Cloud Computing Environment. Computer and Information Security Handbook, 2nd edn. Morgan Kaufmann, San Francisco (2013)
10. Smith, W.E., Trivedi, K.S., Tomek, L.A., Ackaret, J.: Availability analysis of blade server systems. IBM Syst. J. **47**(4), 621–640 (2008)
11. Kim, D.S., Machida, F., Trivedi, K.S.: Availability modeling and analysis of a virtualized system. In: 15th IEEE Pacific Rim International Symposium on Dependable Computing, PRDC 2009 vol., no., pp. 365–371, 16–18 Nov 2009
12. Birke, R., Giurgiu, I., Chen, L.Y., Wiesmann, D., Engbersen, T.: Failure analysis of virtual and physical machines: patterns, causes and characteristics. In: Dependable Systems and Networks (DSN) (2014)
13. Cloud harmony. https://cloudharmony.com/status-1year-of-compute (2015). Accessed 15 May 2015
14. EC2 pricing model. http://aws.amazon.com/ec2/pricing/ (2015). Accessed 15 May 2015
15. Cérin, C., et al.: Downtime statistics of current cloud solutions. In: International Working Group on Cloud Computing Resiliency. https://iwgcr.files.wordpress.com/2012/06/iwgcr-paris-ranking-001-en1.pdf (2015). Accessed 15 May 2015

16. Mohammed, A.B., Altmann, J., Hwang, J.: Cloud Computing Value Chains: Understanding Businesses and Value Creation in the Cloud, Economic Models and Algorithms for Distributed Systems. Birkhäuser, Basel (2010)
17. Gorbenko, A., Kharchenko, V., Mamutov, S., Tarasyuk, O., Romanovsky, A.: Exploring uncertainty of delays as a factor in end-to-end cloud response time. In: 2012 Ninth Euro-pean Dependable Computing Conference, pp. 185–190, IEEE (2012)
18. Ghosh, R., Longo, F., Frattini, F., Russo, S., Trivedi, K.S.: Scalable analytics for IaaS cloud availability. IEEE Trans. Cloud Comput. **2**(1), 57–70 (2014)
19. Helvik, B.E., Gonzalez, A.J.: System management to comply with SLA availability guarantees in cloud computing. In: Proceedings of CLOUDCOM (2012)
20. Yang, B., Tan, F., Dai, Y.-S.: Performance evaluation of cloud service considering fault recovery. J. Supercomput. **65**(1), 426–444 (2013)
21. Cinque, M., et al.: To cloudify or not to cloudify: the question for a scientific data center. In: IEEE Transactions on Cloud Computing (2015)

Stochastic Model-Based Analysis of Energy Consumption in a Rail Road Switch Heating System

Davide Basile(✉), Silvano Chiaradonna, Felicita Di Giandomenico, Stefania Gnesi, and Franco Mazzanti

Consiglio Nazionale delle Ricerche, ISTI-CNR,
Istituto di Scienza e Tecnologia dell'Informazione "A. Faedo", Pisa, Italy
{davide.basile,silvano.chiaradonna,felicita.digiandomenico,
stefania.gnesi,franco.mazzanti}@isti.cnr.it

Abstract. Rail road switches enable trains to be guided from one track to another, and rail road switches heaters are used to avoid the formation of snow and ice during the cold season in order to guarantee their correct functioning. Managing the energy consumption of these devices is important in order to reduce the costs and minimise the environmental impact. While doing so, it is important to guarantee the reliability of the system.

In this work we analyse reliability and energy consumption indicators for a system of (remotely controlled) rail road switch heaters by developing and solving stochastic models based on the Stochastic Activity Networks (SAN) formalism. An on-off policy is considered for heating the switches, with parametric thresholds representing the temperatures activating/deactivating the heating. Initial investigations are carried on to understand the impact of different thresholds on the indicators under analysis (probability of failure and energy consumption).

1 Introduction

A rail road switch is a mechanism enabling trains to be guided from one track to another. It works with a pair of linked tapering rails, known as points. These points can be moved laterally into different positions, in order to direct a train into the straight path or the diverging path. Such switches are therefore critical components in the railway domain, since reliability of the railway transportation system highly depends on their correct operation, in absence of which potentially catastrophic consequences may be generated.

Unfortunately, during winter, snow and ice can prevent the switches to work properly. Indeed the mechanisms which allow a train to be directed can be blocked by an excessive amount of snow or ice. To overcome this issue, the rail road switches need to be cleaned from possible snow or ice forming on top of it. In the past, the switches were kept clear manually by employers who were sweeping the snow away. More recently, heaters are used so that the temperature

A. Fantechi and P. Patrizio (Eds.): SERENE 2015, LNCS 9274, pp. 82–98, 2015.
DOI: 10.1007/978-3-319-23129-7_7

of the rail road switch can be kept above freezing. The heaters may be powered by gas or electricity.

The managing of the heaters is automatic, and is controlled by a central unit using a Powerline [3]. Powerline communications have the possibility to transmit coded information through the existing electric lines. A Powerline is a transmission system that uses electric lines, with a very extensive infrastructure in nearly each building. The main advantage of adopting the existing network is the absence of additional costs for the installation of the infrastructure.

Powerlines are used in order to automate and optimize several tasks, i.e. switching on and off the lights of the station, checking the status of railway track switches, managing the traffic of trains in the station and the rail road switch heating system.

Nowadays, there is a great attention towards cautious usage of energy sources to be employed in disparate application domains, including the transportation sector, to save both in financial terms and in environmental impact. Therefore, studies devoted to analyse and predict energy consumption are more and more gaining importance, especially in combination with other non functional properties, such as reliability, safety and availability.

In this paper, we address reliability and energy consumption of rail road switch heaters. A failure in those switches can lead to major malfunctions. Indeed, while managing energy optimization we must ensure reliability. In fact, by turning on all the heating system at the same time, an overhead of energy consumption can lead to a blackout. Alternatively, an excessively parsimonious policy to save on energy can cause the failure of some rail road switch heaters. The proposed analysis contributes to gain insight on the interplay between energy consumption and reliability in order to select an appropriate policy for the heating of the switches by selecting minimum and maximum temperature thresholds, which guarantees a satisfactory trade-off. Note that failure of the heating system is accounted for by other components of the railway system, namely interlocking mechanisms which guarantee safety; however we do not include them in our analysis.

We adopt a stochastic model-based approach to analyse the behaviour of the rail road switch heating system under different circumstances. A modular and parametric approach is followed, to assure usability of the developed analysis framework in a variety of system configurations, as well as to promote extension and refinements of the model itself, to account for further involved aspects/phenomena and so enhance its adherence to sophisticated and realistic implementations with respect to the current preliminary version. In particular, we exploit *Stochastic Activity Network* (SAN) [15] to model the heating system, and use the Möbius tool [4] to perform experiments.

Structure of the Paper. In Sect. 2 we describe the considered rail road switch heating system. Stochastic model-based analysis is introduced in Sect. 3, while in Sect. 4 we present the models of the rail road switch heating system. The results of our experiments are discussed in Sect. 5, related work and conclusions are in Sects. 6 and 7.

2 The System Under Analysis

In Fig. 1 a rail road switch heater is displayed. The picture is taken from [14].

The heating system consists in a series of tubular flat heaters along the rail road track, which warm up the rail road by induction heating. To accomplish its task, the rail road switch heater system reads through sensors the temperatures of the air and of the rail road, and checks if the temperatures are between given thresholds [14]. Based on this general behaviour, we have based the policy employed to activate/deactivate the heating on two threshold temperatures:

- *warning threshold*: this temperature represents the lower temperature that the system should not trespass. If the temperature is lower than the warning threshold, then the risk of ice or snow can lead to a failure of the rail switch and therefore the heating system needs to be activated;
- *working threshold*: this is the working temperature of the heating system. Once this temperature is reached, the heating system can be safely turned off in order to avoid an excessive waste of energy.

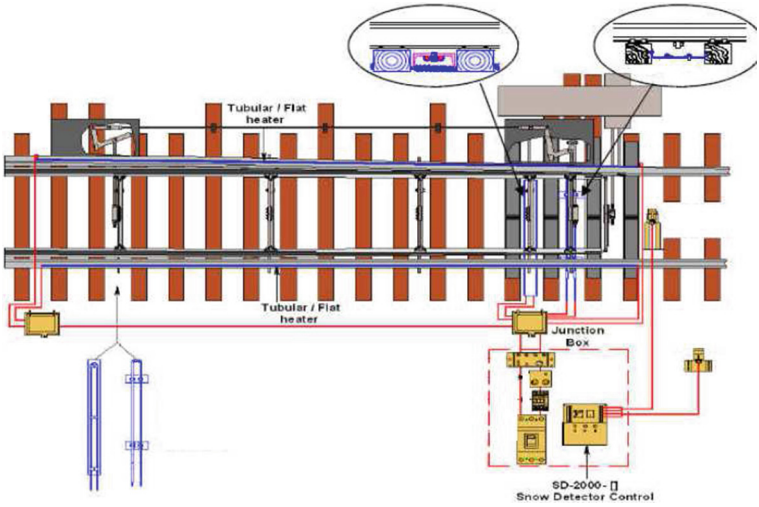


Fig. 1. An example of a rail road switch heater heathen by electric induction ©2006 – 2012 Rails Company

Hence the time during which a single heater is active depends on its location and the weather conditions. Heaters located in a colder geographical area will consume more energy then those under warmer temperatures.

The network of heaters will share information through central computational unit, using the Powerline. The central unit manages the maximum amount of power that can be delivered to the system, in order to prevent possible blackouts.

3 Stochastic Model-Based Analysis

Stochastic model-based approaches are useful to support the development of new systems, in all the phases of their life cycle. In the early design phases it is important to validate a model of a system in order to avoid waste of time and resources in the development phase. This can be done by pointing out the properties and the requirements of the system, building a model that represents its behaviour and checking that the properties are satisfied by the model. It is possible to choose between different alternatives for the same system, and select the one that better suits the requirements. An early modelling phase is also useful to highlight problems in the design of the system.

When the design phase is completed, a model allows predicting the overall behaviour of the system, fostering an analysis for the fulfilment of constraints in the design phase and the acceptance cases. For an already existing system, an a-posteriori analysis of properties such as dependability or performance is useful in order to improve the system in its future releases. Moreover, with a model-based analysis it is possible to predict future behaviour to plan the maintenance and the upgrading of the system.

3.1 Stochastic Activity Network

Several stochastic modelling methodologies have been proposed in literature. Stochastic Activity Networks [15] are a widely adopted formalism for the analysis of systems under performance, dependability and quality of service. The formalism is a generalization of Stochastic Petri Nets [2], and has similarities with Generalised Stochastic Petri Nets [1]. A SAN is composed of the following entities: *places, activities, arcs, input gates* and *output gates*. Places in SAN have the same interpretation as those of Petri Nets. Activities are of two types: *instantaneous* and *timed*. Instantaneous activities are fired once the enabling conditions are satisfied. Timed activities are fired following a temporal stochastic distribution of time. There are different policies of activation and reactivation of timed activities, for a marking based policy of reactivation of timed activity. An enabled activity is aborted when the SAN moves into a new marking in which the enabling conditions of the activity no longer hold. Cases are associated to activity, and are used to represent uncertainty about the action taken upon completion of the activity. A marking is stable if no instantaneous activity is activated. Input gates control the enabling of the activity and the change of marking at completion of the activity. Output gates define the change of marking upon completion of activity. The primitives of the SAN (activities, input and output gates) can be defined using C++ code. When an activity completes, the following sequence of events is executed:

- one of the cases of the activity is chosen according to its probability,
- the functions of the connected input gates are executed,
- one token is removed from the places connected by the input arc,
- the function of the output gates connected to the activity are executed,

– one token is added to the places connected to the activity or one of its cases by the output gate.

For evaluating the energy consumption and the probability of failure of the system modelled, we use the Möbius tool [4]. Möbius is a software tool for modelling the behaviour of complex systems, supporting various formalisms such as SAN, PEPA, Fault Tree, etc... Developed models are then solved by using different analytical and simulative solvers. This tool can be used for studying the reliability, availability, and performability of systems. It follows a modular modelling approach, with proper operators *Rep* and *Join* to compose atomic models into an overall composed model.

4 Modelling Framework

In this section we describe the rail road heating system model.

We developed a SAN model to represent the system which is actually structured as composed by five atomic sub models, properly combined through Rep and Join operators (see Fig. 4).

Three of the five atomic models are selectors for the profile, locality and the unique identifier of each switch. The remaining two are the atomic model for the queue, shared among the replicas, and the main model for the rail road switch heater.

The main parameters of the SAN model are the lower and working temperature of the device, and the maximum power that the system can provide every instant of time, i.e. the maximum number of heaters that can be turned on at the same time.

Weather forecast. To model the external weather conditions, our model takes in input a table containing profiles of average temperatures in those days for which the analysis is relevant (a.g. winter days).

The time window under analysis is divided in intervals to which an average reference temperature is assigned. Current instance of the model concentrate on nights only, from 6:00 am to 6:00 pm, divided in intervals of two hours. However, the model can be easily modified to consider longer periods, as well as different number of intervals.

At starting time, a SAN will select one of these profiles. A probability is assigned to each profile. Less frequent profiles will have a lower probability. The probability in which a particular profile will be selected, depends on the aforementioned value. In Fig. 2 left, the SAN model corresponding to the action of selecting the profile is displayed. The probability is selected in the SAN model with a timed activity with different cases, each case corresponds to a profile. A C++ function in the output gate will load the selected profile.

We note that an alternative way of deciding the actual temperature would be to select stochastically the temperature depending on the previous temperature and the actual time. However, by adopting this methodology we would have obtained a non realistic zig-zag evolution of the temperature during the

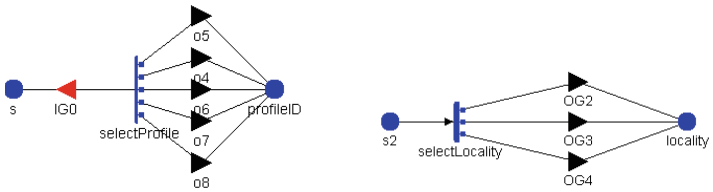


Fig. 2. On the left the SAN model ProfileSelector, on the right the SAN model LocalitySelector.

night. Indeed after each interval temperatures would have a non-zero probability of increasing or decreasing. Thus there could be the unwanted case where temperatures increase and decrease several times during the night.

Location. Rail road switch heaters are located in different zones, with different weather conditions, while we assume that they are remotely controlled by the central management unit. At a higher altitude, the probability to have temperatures lower than the average selected for that profile is greater than the case of rail road switch heaters located at lower altitudes. For modelling this variation of temperature, the model takes as input a vector of locations, which are numerical values representing the gap from the average temperature for that profile. In case there are many rail road switch heaters located at higher altitudes, it will be more probable that the actual temperature will be lower than the average. The SAN that selects the location for the device is depicted in Fig. 2 right. The network works similarly to the one for the selection of the profile. A timed activity has different cases, according to the different probabilities of selecting a location. Once the activity is fired, a C++ function is called which assigns the given probability to the rail road switch heating module.

4.1 Rail Road Switch Heater

In Fig. 3 the SAN model representing the rail road switch heater is depicted. The SAN takes in input the profile and the location of the device, and a unique identifier. This unique identifier will be computed by a separate SAN, not shown here. There are two subnets present in the network:

Heater subnet represents the status of the heater. It can be switched on or off, or it can be a failure state. An extended place *temperature* represents the internal temperature of the device, while the places *on* and *off* represent the status of the system. A shared place *sharedOn* is useful to know how many heaters are turned on at a given time. According to the heating policy, once the system temperature goes below a pre-defined *warning threshold*, the heating needs to be activated otherwise a switch failure is experienced (represented by the place *failure*). Then, once the temperature raised and reached the *working threshold*, the heating system can be safely turned off.

The energy consumption of the overall system depends on the value of those thresholds. A smaller gap between the two values represents a frequent activation of the heating system, but for a shorter period of time. Alternatively, by taking a wider gap between the two values, we will obtain a less frequent activation, but the activation will be for longer periods of time. In the next section, we instantiate the model with varying values for these thresholds, to show their impact and benefit on both energy consumption and system reliability.

Clock subnet represents a clock which updates at each unit of time the parameters, i.e. the temperature of the rail road and the temperature of the air. For this case study, the unit of time is assumed to be one hour. Each time the clock activity is fired, the temperatures will be updated in the output gate using C++ functions. We model the physical behaviour of the rail road in terms of temperature decay and increase, when the heating is switched off and on, respectively. For the temperature of the air, a new value will be picked up by the table for the selected profile and depending on the location of the device, this value will be higher or lower than a given amount of degrees, selected in the locality table.

For the internal temperature of the device, we will model the actual temperature of the system by using an equation of heat exchange through convection, which simulates the data that will come as input from the sensors.

The *energy* place represents the amount of energy consumed by the system. Each time the clock activity is fired, if the heater is switched on, then the energy consumed is augmented. By taking into account the power consumed by the heater, it is possible to calculate the consumption of energy in kilowatt.

Moreover, in Fig. 3, the place *free* represents the queue of active rail road switch heaters. This place is shared among all the instances of the SAN. The capacity of the queue represents the number of heaters that can be turned on at the same time, that we call NH_{Max}. For example, assuming that the power consumed by a single heater per hour is $35KW$, if $NH_{Max} = 2$ than the maximum energy that the system can provide is $70KW$: no more than two heaters can be turned on at the same time. If the capacity of the queue is exceeded, a blackout failure may occur. Hence the thresholds for the activation of the switches must be chosen also taking into account this constraint. With the queue model, it is also possible to implement different priorities in which a particular rail road switch heater must be turned on and off (not implemented in this paper). Although a sub-model *Queue* is included in the composed model in Fig. 4, in this work it is only consists of the place *free*.

4.2 Physical Model for Heat Exchange

As mentioned above, we need to simulate the data read by the sensors in order to estimate the energy consumption of the heaters. To do so, we instantiate a physical model representing the exchange of heat through convection. Indeed the rail road gets cooled by the external temperature and warmed by the heaters.

To make the needed calculation we consider the portion of the rail road track to be heated, which for simplicity is an iron bar representing the rail road track.

We assume that the bar is exposed to the external temperature both from the top and the bottom.

The heater is represented by an electric cable that passes through the rail road in different points in order to warm up the iron. We assume that the power used by the heater is constant, in order to estimate the kilowatt per hours consumed during the time interval that we consider (6pm - 6am).

Every hour the sensor reads a new data for the internal temperature of the rail road track. Assuming that the value of the temperature of the air and the previous internal temperature are known, we foresee the updated internal temperature of the device using the following equation representing exchange of heat by convection. This equation is derived from a differential equation on the time:

$$T_{fin} = T_a + (T_i - T_a) \cdot e^{\frac{-u \cdot A \cdot t}{m \cdot c}} + \frac{Q}{u \cdot A \cdot L}$$

The coefficient of convective exchange u is calculated as:

$$u = (\frac{g \cdot \beta \cdot (T_i - T_a) \cdot \rho^2 \cdot AV^3}{\mu^2})^{\frac{1}{4}} \cdot \frac{0.54 + 0.26}{2} \cdot \frac{K}{AV}$$

The parameters of the previous equation are: T_{fin} is the new internal temperature of the heater; u is the the coefficient of convective exchange; c is the heat capacity of iron; A is the surface area exposed to the external temperature; AV is the ratio between area and volume of the iron bar; t is the interval of time, one hour in our case; m is the mass of the iron bar; T_i is the previous internal temperature; T_e is the external temperature of the surrounding area; Q is the power used when the heater is turned on, if the heater is turned off this value will be zero. Moreover L is the length of the electric cable for heating the rail road; g is the gravity acceleration; β is the thermal expansion coefficient; ρ is the density of air; μ is the dynamic viscosity and K is the thermal conductivity of iron.

The function representing the heating exchange is written in C++, and it is called by the output gate $O1$ displayed in Fig. 3 to update the temperature of the rail road every hour. This C++ module is the portion of the system representing the real world model that we want to optimize. This module can be easily modified to reflect different scenarios of energy optimization. For example, energy optimization could be applied to the enlightening of a station, by turning off the lights when they are not needed. In this case, the temperature should represent the quantity of light during that part of the day, the thresholds should represent when the lights must be turned on and off, while the energy consumption equation should take into account the amount of consumed electricity.

4.3 The Composed Model

The composed model is displayed in Fig. 4. The atomic SAN models are *RailSwitchHeater, LocalitySelector, ProfileSelector, SwitchIDSelector* and *Queue*. The box *Join*1 represents the join of all the atomic models except *Queue*. Those

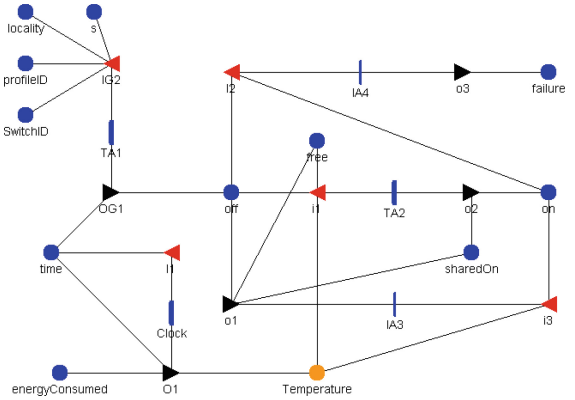


Fig. 3. The SAN model RailRoadSwitchHeater.

atomic models share the places relative to the locality of the device, its weather profile and unique ID. The box *Rep*1 represents the network of heaters. A parameter *numRep* identifies the number of devices composing the network. Recall that each device has its own weather forecast profile and locality. Finally, the node *Join*2 composes the network of rail road switch heaters with the model representing the Queue. Indeed all the sub models share the same queue.

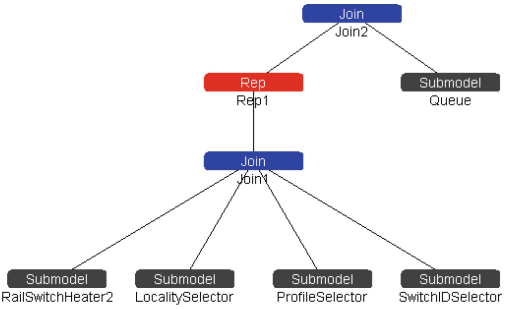


Fig. 4. The composed model

5 Analysis results

In this section we describe the preliminary experiments we have performed in order to find suitable trade-offs in terms of reliability and energy consumption for different settings of the model parameters. These experiments do not refer to a rail road configuration taken from reality, but consider a restricted plausible configuration adopted as a working example to show the benefits of our

analysis framework. Indeed, the parametric nature of our model allows its customization to a wide variety of system configurations in terms of both size and characterization of system components.

5.1 Measures of Interest

We consider two different measures of interest which represent the energy consumption and reliability of the system under analysis:

1 $CE(t, l)$: the mean energy consumed by a heater in the interval $[t, t + l]$;
2 $PFAIL(t, l)$: the mean probability that a switch fails (becomes frozen) at time $t + l$, given that at time t is not failed.

The interval $[t, t + l]$ goes from 6:00 pm to 6:00 am. $CE(t, l)$ is defined by accumulating in the interval $[t, t+l]$, that is the time that each replica of the SAN model RailSwitchHeater spends in the marking represented by one token in the place ON, multiplied for the energy consumed in a unit of time. $PFAIL(t, l)$ is defined as the probability that at time $t+l$ there is one token in the place Failure of the SAN model RailSwitchHeater. $PFAIL(t, l)$ can be used to compute the mean time to a catastrophic failure.

Other measures may help in understanding aspects related with energy consumption and system failure, so to take appropriate actions in improving the system. Some of them are introduced in the following, although their quantitative evaluation is left as future work:

- the probability of heating activation in an interval of time; it is useful for understanding in which hours the probability that a heater is switched on is greater, i.e. the hours of the night with a greater consumption of energy;
- the probability of activation of a given number of heaters in an interval of time; it is useful for improving planning operations performed by the central computational unit;
- the average number of heaters switched on in an interval of time; again, it is useful for planning purposes.

5.2 Scenarios and Settings

The table of the profiles used for the experiments are displayed in Table 1. We consider average cold winter nights, which are primary relevant for our study. Starting from the temperature at 6 pm, it decreases and reaches the minimum at 6 am.

We consider three main localities, displayed in Table 2. We note here that it is more probable to select a locality at a lower altitude, where there is no variation between the selected profile temperature. It must be pointed out that these parameters can be easily modified to deal with different conditions.

The parameters considered for calculating the heat exchange between the rail road, the heaters and the external air are showed in Table 3. We consider an iron bar of 6 meters of length, 4 cm of height and 28 cm of width to represent

Table 1. The profiles of temperatures per night

	06 pm	08 pm	10 pm	00 am	02 am	04 am	06 am	Probability
profile 1	2°	0°	-2°	-2.5°	-3°	-3.3°	-3.8°	0.25
profile 2	0°	-1°	-2°	-3°	-4°	-4.5°	-5°	0.2
profile 3	3°	1°	0°	-1°	-2°	-3°	-3.5°	0.2
profile 4	0°	-2°	-3.5°	-4.8°	-6.2°	-6.8°	-6.9°	0.15
profile 5	1°	-1°	-3°	-3.5°	-4°	-4.5°	-5.5°	0.2

Table 2. The temperature variation and the probability of occurrence of such variation for the three considered localities

	Temperature Variation	Probability
locality 1	-0.8 °	0.3
locality 2	-0.3 °	0.3
locality 3	0 °	0.4

the portion of rail road track that needs to be heated. In the table l, h and w are respectively the length, height and width of the iron bar. The other parameters are described in Sect. 4, and we use standard value for the parameters of the air and of the iron, i.e. viscosity, density, conductivity, heat capacity etc... We assume that all the rail road switch heaters use the same amount of power, which in this case is 35 KiloWatt.

Table 3. The parameters for the model of heat exchange by convection

l	h	w	T_i	β	ρ
6 m	28 cm	4 cm	5 °	0.0037 C^{-1}	1.30 $\frac{Kg}{m^3}$

μ	K	AV	A	m	c	Q
17.157 $\cdot 10^{-6} Pa \cdot sec$	0.023 $\frac{W}{mK}$	0.0175 m	1.68 m^2	529.13 Kg	450 $\frac{J}{KgK}$	35 KW

For this case study, we set the freezing thresholds to zero. If the temperature goes below this value the system reaches a failure state. In our experiments we consider four different combinations of thresholds, at different temperatures and with tight or wide gaps between the working threshold and the warning threshold.

The network is composed of four rail road switch heaters, and we study the results at varying of NH_{Max}, representing how many heaters can be turned on at the same time, that in turn represents the maximum throughput of energy of the system in a unit of time. We assume that all the heaters have initial temperature of 5 degrees Celsius.

The four pairs of thresholds are (1,4), (4,7), (5,6), (4,50). We consider different gaps between the thresholds. Moreover we study the behaviour of the heating system for thresholds that are more and more distant from the temperature of failure.

The experiments were taken on the Möbius tool [4], by using simulation with 10000 batches. The results were computed in one minute on a machine with processor Intel Core i3.2328M 2.20 GHz with 8 GB of RAM, showing the efficiency of the model.

5.3 Results Discussion

We report the results for two of the aforementioned measures of interest: $CE(t, l)$ and $PFAIL(t, l)$.

The results for the probability of failure are reported in Fig. 5. As expected, by minimizing the throughput of energy the probability of failure increases.

Moreover, the probability of failure increases in case the gap between the two thresholds is greater. Indeed the time taken by the heater to warm up the rail road is higher when the gap is wider. Hence the temperature of the other pending heaters may go below the freezing threshold, leading to a failure of the system.

The results for the energy consumption are reported in Fig. 6. We note that by increasing NH_{Max} we have more energy available, which in turn results in a major energy consumption (recall Sect. 4). The same happens when we consider greater value for the thresholds.

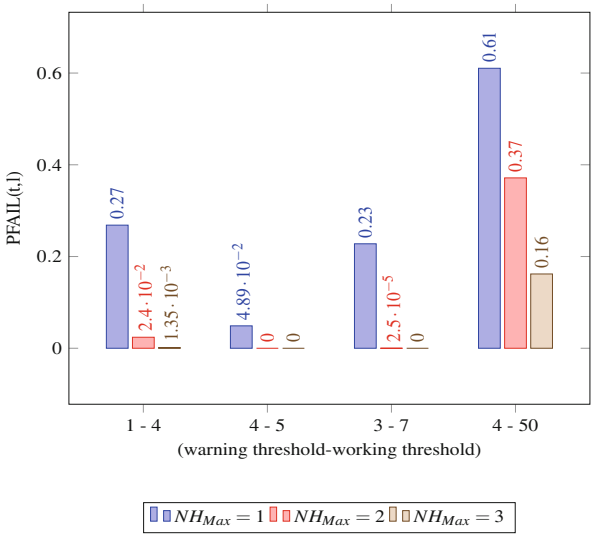


Fig. 5. The graph for the probability of failure

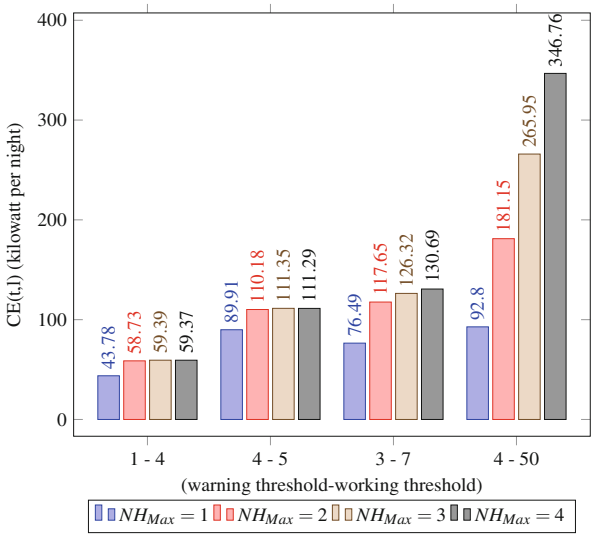


Fig. 6. The graph for the energy consumption

Interestingly, a wider gap is worse than a strict gap when NH_{Max} is large enough, while the converse holds in case NH_{Max} is limited.

The threshold $(4, 50)$ is used for checking the consumption of energy in case the heater, once activated, will never be turned off during the whole night. Indeed at 35 KW of power, the system never reaches the working temperature. Of course this threshold is the worst both for the probability of failure, and for the energy consumption. Note that if the starting temperature is not below the warning threshold, the heater will not be activated for the whole night. Indeed the temperature of the heater needs some hours to fall below the warning threshold. For example, in Fig. 6 for thresholds $(4, 50)$ and $NH_{max} = 1$ we have $CE(t, l) = 92.8 \approx \frac{10.5*35}{4}$, so after two or three hours the temperature of the track will go below 4 degrees Celsius.

Our experiments show that it is better to take a tight gap between the minimum and the maximum thresholds. Also keeping those thresholds not too close to the freezing threshold improves reliability, as well as guarantees a sufficient amount of heaters to be activated at the same time. Indeed this is necessary for guaranteeing the reliability of the system and minimizing the energy consumption.

Hence, the best scenario in this example is represented by the pair of thresholds $(1, 4)$ for $NH_{Max} = 3$ and $(4, 5)$ for $NH_{Max} = 2$.

6 Related Works

Although not tailored to the rail road switch heating system, there are several works in the literature that analyse and optimise the energy consumption in

several application domains using formal approaches. A few of them are recalled in the following.

In [13] Generalized Stochastic Petri Nets [1] are used to solve the dynamic power management problem for systems with complex behaviour. Dynamic power management addresses reduction of power dissipation in embedded systems, with a selective shut-off or slow-down of system components that are idle or underutilized. A time-out policy is used for power saving, which turns on a component when it is used and turns it off when it is not used for a certain amount of time. Comparisons are also performed with other models based on Markov Decision Processes (MDP). GSPN allows to express a finer model, with synchronizations and conflicts between different modules, that is shown to be more accurate in power saving then MDP models. In our case complex behaviours are modelled with SAN, which are a generalization of GSPN. We also consider a policy of switching on/off the heater when a given temperature threshold is reached. We express a finer behaviour by using C++ code in the SAN model which computes the physical model of the heat transfer. Indeed the amount of time in which a heater is not used, (i.e. turned off) is derived from the external temperature, the internal temperature of the iron bar, and the heat transfer law.

In [11] the problem of power management in smart grids is handled with Learning Automata (LA), that provide a mechanism of learning from the environment the optimal solution over a period of time. The model of the system is hierarchical: at the root there is a LA-based main power station, that supplies the power to LA-based transmission system, and adjusts the power supply according to requirements based on learning the system. The LA-transmission system calculates the performance of the system. The studied performance metrics are the power utilization and the customer satisfaction, in terms of satisfied energy demand. It is shown that, by adjusting the power supplied to the different clients, it is possible to obtain a good trade-off between power utilization and customer satisfaction. In [7] the dynamic power management problem is interpreted as a hybrid automaton control problem and integrated stochastic control. Hybrid automata mixed both a discrete state, representing the power mode of the system, and a continuous one, representing the consumed power. An integrated stochastic control is synthesised based on a learning feedback, and it is used to predict probabilistically the range lengths of the future idle period based on the past history. Two strategies are compared: on demand wake-up of a component (that was previously turned off) and pre-emptive wake-up. The former provides better results for conservation of energy and prevention of latency. In our work we do not implement learning mechanisms, and the power supplied is fixed by the number of heaters that can be turned on in a unit of time. It would be interesting to relax this constraint and implement a power adjustment with a prediction mechanism for minimising the power supplied, for example in case of warmer nights.

In [12] the applicability of self-organizing systems for different fields of power system control is discussed. Agent-based decentralized power flow control is com-

pared with current practice based on central decision making. The authors study how to balance the voltage and frequencies stability of the network to meet the demand of energy. These parameters are linked to reliability and safety of the system. It is shown how a decentralized control can improve reliability, safety and efficiency by providing a real-time adaptivity to changes in the network (failure of a node, blackout). In our case we consider a central unit which manages the different heaters. The demand of energy is adjusted according to the maximum energy that can be delivered by the central unit. In case of failure of a heater, the energy is automatically shared between the remaining active heaters. We show that by managing the temperature thresholds it is possible to improve reliability even in case of low energy demand.

In [8] the authors analyse the survivability of a smart house, that is the probability that a house with locally generated energy (photovoltaic) and a battery storage can continuously be powered in case of a grid failure. Hybrid Petri Nets [5] are used for modelling this scenario. Different strategies of battery management are considered. In the first all the battery is consumed when needed, in the second there is a minimum threshold of energy saved in case of grid failure. In the third case the battery is also charged to a maximum threshold when the grid is operating. It is shown how the third strategy is better both for the local usage of energy and for the survivability of the smart house. The authors consider a randomly chosen probability of failure and fixed thresholds. Instead, in our case the probability of failure is derived from the model. Moreover, we do not consider fixed thresholds, but we analyse how different values for thresholds impact in the energy consumption and reliability.

Concerning the analysis and optimization of a railway station using formal techniques, in [6] Stochastic Activity Network are used to improve timetable and delay minimization of the traffic in a station. The model takes in input the railway topology and the required service. Experiments are taken to measure the capacity of the line in terms of number of trains that traverse the line, and the percentage usage of each track segment. In [10] an Automatic Train Supervision is designed that prevents the occurrence of deadlocks. A formal model that designs railway layout and the Automatic Train Supervision behaviour is used to verify such deadlock properties. The verification phase is performed by using the UMC model checking verification framework [9]. It would be interesting to integrate such studies with the possible failure of switches studied here, in order to analyse how a failure in a switch impacts on possible delays of trains, and deadlocks.

7 Conclusion and Future Work

We have presented the result of a preliminary research activity in model-based analysis for a rail road switch heating system. We used Stochastic Activity Networks to evaluate both the energy consumption and the probability of failure.

The system reads in real time both the temperature of the external air, the one of the rail road track, and according to given thresholds decides when to turn

on and off the heaters. We evaluated the probability of failure of the system and the energy consumption at varying the thresholds and the maximum number of heaters that can be turned on at the same time. To represent the heat exchange between the portion of the rail road track, the external air and the heaters, we describe a physical model of heat exchange by convection. Simulations of the model have been taken using the Möbius tool [4]. At the moment we have instantiated the model to representative (but realistic) parameters, to obtain a first indication of the behaviour of our system. The results suggest that, in order to have a good trade off between probability of failure and energy consumption it is important to:

- guarantee enough energy to allow at least two heaters working simultaneously;
- keep the minimum temperature distant from the temperature of failure;
- use a small gap between the minimum and the maximum temperatures, in order to better distribute the time in which each heather is turned on.

This work represents a first step in the design of a rail road switch heating system with attention to energy consumption and reliability, which often are opposed requirements. Several directions for extending this study have been identified.

We plan to instantiate our model to case study from real world, i.e. an existing rail road switch heating system, to confirm our initial investigation.

Additional measures of interest, such as those discussed in Sect. 5, would help to obtain more insight on the behaviour of the system and its energy consumption.

Moreover we plan to study how the energy consumption is modified by changing parameters of the underlying physical model. Indeed, the obtained results may suggest that, by changing the material of which the heaters are composed, its length or the power consumed, a better trade off between reliability and energy optimization can be obtained.

It would also be interesting to let the power consumed by the system vary at different weather conditions. This may help to improve the reliability of the system. Indeed in case of emergency a major throughput may prevent a failure.

Moreover, by implementing a priority queue, it is possible to develop strategies for activating and deactivating those heaters which are in a critical situation, i.e. when the temperature is closer to the failure point. At the moment, in case of conflicts between different heaters that need to be turned on, the choice is made stochastically by the system.

We plan to adapt the model to other different case studies for energy optimization in the context of a railway station, exploiting the modularity of the proposed approach.

Acknowledgements. This work has been partially supported by the projects TENACE (PRIN n.20103P34XC) and CINA (PRIN n.2010LHT4KM), both funded by the Italian Ministry of Education, University and Research. Moreover, we are thankful to Gianpaolo Roina for the useful comments.

References

1. Balbo, G.: Introduction to generalized stochastic petri nets. In: Bernardo, M., Hillston, J. (eds.) SFM 2007. LNCS, vol. 4486, pp. 83–131. Springer, Heidelberg (2007)
2. Bause, F., Kritzinger, P.S.: Stochastic petri nets: an introduction to the theory. SIGMETRICS Perform. Eval. Rev. **26**(2), 2–3 (1998)
3. Berger, L.T., Schwager, A., Joaquín Escudero-Garzás, J.: Power line communications for smart grid applications. J. Electr. Comput. Eng. **2013**, Article ID 712376, 16 (2013). doi:10.1155/2013/712376
4. Clark, G., Courtney, T., Daly, D., Deavours, D., Derisavi, S., Doyle, J.M., Sanders, W.H., Webster, P.: The möbius modeling tool. In: Proceedings of the 9th International Workshop on Petri Nets and Performance Models, pp. 241–250 (2001)
5. David, R., Alla, H.: On hybrid petri nets. Discrete Event Dyn. Syst. **11**(1–2), 9–40 (2001)
6. Di Giandomenico, F., Fantechi, A., Gnesi, S., Itria, M.L.: Stochastic model-based analysis of railway operation to support traffic planning. In: Gorbenko, A., Romanovsky, A., Kharchenko, V. (eds.) SERENE 2013. LNCS, vol. 8166, pp. 184–198. Springer, Heidelberg (2013)
7. Erbes, T., Shukla, S.K., Kachroo, P.: Stochastic learning feedback hybrid automata for dynamic power management in embedded systems. In: SMCia/05, IEEE Mid-Summer Workshop on Soft Computing in Industrial Applications, June 2005
8. Ghasemieh, H., Boudewijn, R., Haverkort, M.R.J., Remke, A.: Energy resilience modeling for smart houses. In: DSN (2015, to appear)
9. Gnesi, S., Mazzanti, F.: An abstract, on the fly framework for the verification of service-oriented systems. In: Wirsing, M., Hölzl, M. (eds.) SENSORIA. LNCS, vol. 6582, pp. 390–407. Springer, Heidelberg (2011)
10. Mazzanti, F., Spagnolo, G.O., Della Longa, S., Ferrari, A.: Deadlock avoidance in train scheduling: a model checking approach. In: Lang, F., Flammini, F. (eds.) FMICS 2014. LNCS, vol. 8718, pp. 109–123. Springer, Heidelberg (2014)
11. Misra, S., Krishna, P.V., Saritha, V., Obaidat, M.S.: Learning automata as a utility for power management in smart grids. IEEE Commun. Mag. **51**(1), 98–104 (2013)
12. Müller, S.C., Häger, U., Rehtanz, C., Wedde, H.F.: Application of self-organizing systems in power systems control. In: Dieste, O., Jedlitschka, A., Juristo, N. (eds.) PROFES 2012. LNCS, vol. 7343, pp. 320–334. Springer, Heidelberg (2012)
13. Qiu, Q., Wu, Q., Pedram, M.: Dynamic power management of complex systems using generalized stochastic petri nets. In: DAC, pp. 352–356 (2000)
14. http://www.railsco.com/~electric_switch_heater_controls.htm
15. Sanders, W.H., Meyer, J.F.: Stochastic activity networks: formal definitions and concepts. In: Brinksma, E., Hermanns, H., Katoen, J.-P. (eds.) EEF School 2000 and FMPA 2000. LNCS, vol. 2090, pp. 315–343. Springer, Heidelberg (2001)

Bidirectional Crosslinking of System and Software Modeling in the Automotive Domain

Harald Sporer(✉), Georg Macher, Andrea Höller, and Christian Kreiner

Institute of Technical Informatics, Graz University of Technology,
Inffeldgasse 16/1, 8010 Graz, Austria
{sporer,georg.macher,andrea.hoeller,christian.kreiner}@tugraz.at
http://www.iti.tugraz.at/

Abstract. Replacing former pure mechanical functionalities by mechatronics-based solutions, introducing new propulsion technologies, and connecting cars to their environment are only a few reasons for the still growing E/E-System complexity at modern passenger cars. Hence, for an engineering company in the automotive embedded system domain it is vital to establish mature development processes, including a smart tool chain orchestration. Starting from the customer requirements until the final release of the product, traceability and consistency between all development artifacts shall be given. However, achieving this by linking the development items manually is a tedious and error-prone task. The aim of this work is to enhance the development process by introducing a fully automatic transformation of a system design model into a software framework model and vice versa. With this novel approach, the full traceability, between the system and software architectural levels, is guaranteed.

Keywords: Automotive · Model-based development · Embedded systems · Traceability · Model-based software engineering

1 Introduction

Embedded systems are already integrated into our everyday life and play a central role in all domains including automotive, aerospace, healthcare, industry, energy, or consumer electronics. In 2010, the embedded systems market accounted for almost 852 billion dollar, and is expected to reach 1.5 trillion by 2015 (assuming an annual growth rate of 12 %) [17]. Current premium cars implement more than 90 electronic control units (ECU) with close to 1 Gigabyte software code [6], are responsible for 25 % of vehicle costs and an added value between 40 % to 75 % [22].

The trend of replacing traditional mechanical systems with modern embedded systems enables the deployment of more advanced control strategies providing additional benefits for the customer and environment, but at the same

A. Fantechi and P. Patrizio (Eds.): SERENE 2015, LNCS 9274, pp. 99–113, 2015.
DOI: 10.1007/978-3-319-23129-7_8

time, the higher degree of integration and criticality of the control application raise new challenges. To cope with this situation, smart methods and techniques have to be applied starting from the very beginning of a systems development. Some kind of guidelines, like the functional safety standard for E/E-Systems at modern passenger cars ISO 26262, has been introduced in recent years.

To handle upcoming issues with modern real-time systems, also in relation to ISO 26262, model-based development supports the description of the system under development in a more structured way. Model-based development approaches enable different views for different stakeholders, different levels of abstraction, and central storage of information. This improves the consistency, correctness, and completeness of the system specification and thus supports the demands of time-to-market (first time right). Nevertheless, such seamless integration of model-based development are rather exception than the rule and often fall short due to the lack of integration of conceptual levels and tooling levels [3].

The aim of this paper is to bridge the existing gap between model-driven system engineering tools and software engineering tools. More specifically, the approach is based on the enhancement of a model-driven system-engineering framework with software-architecture design capabilities. Furthermore, a model-transformation framework enables a seamless description of safety-critical software, from requirements at the system level down to software component implementation in a bidirectional way. The model-transformation framework automatically generates software architectures in Matlab/Simulink described via high level control system models in SysML format. The goal is, on one hand, to support a consistent and traceable refinement from the early concept phase to software implementation. On the other hand, the bidirectional update function of the transformation framework enables facilitation of gaining mutual benefits for basic software and application software development from the coexistence of both information within the central database.

The document is organized as follows: In the course of this paper, Sect. 2 presents an overview of related approaches as well as model-based development and integrated tool chains. In Sect. 3 a description of the proposed bridging approach for the refinement of the model-based system engineering model to software development is provided. An application and evaluation of the approach is presented in Sect. 4. Finally, this work is concluded in Sect. 5 with an overview of the presented approach.

2 Related Works

Model-based systems and software development, as well as tool integration are engineering domains and research topics aimed at moving the development steps closer together and thus improving the consistency of the system over the expertise and domain boundaries. In Pretschner's roadmap [18], the authors highlight the benefits of a seamless model-based development tool-chain for automotive software engineering.

Broy et al. [3] mention concepts and theories for model-based development of embedded software systems. The authors claim model-based development the

best approach to manage the large amount of information and complexity of modern embedded systems with safety constraints. The paper illustrates why seamless solutions have not been achieved so far, they mention commonly used solutions, and arising problems by using an inadequate tool-chain (e.g. redundancy, inconsistency and lack of automation).

Nevertheless, the challenge of enabling a seamless integration of models into model-chains is still an open issue [19,20,24]. Often, different specialized models for specific aspects are used at different development stages with varying abstraction levels. Traceability between these different models is commonly established via manual linking due to process and tooling gaps. Holtmann et al. [8] claim this lack of automation for those linking tasks and missing guidance which model should to be used at which specific development stage as crucial drawback of model-driven development (MDD). The very specific and non-interacting tools requiring manual synchronization, are often inconsistent or rely on redundant information.

An issue is also addressed by Giese et al. [7]. System design models have to be correctly transferred to the software engineering model, and later changes must be kept consistent. The authors propose a model synchronization approach consisting of tool adapters between SysML models and software engineering models in AUTOSAR representation. A drawback of this approach, each transformation step implies potential sources for ambiguous mapping and model mismatching.

An important topic to deal with is the gap between system architecture and software architecture - especially while considering component-based approaches such as UML and SysML for system architecture description and AUTOSAR for SW architecture description. Using SysML [2,7,10,12,15] or X-MAN [11] for architectural description and AUTOSAR for software system description are two common variants in the automotive domain. Buchmann et al. [4] present an approach of another domain, based on bi-directional and incremental transformation between XML class diagrams and Java source code. The authors also highlight the crucial importance of powerful tool support for model-driven software engineering. Boldt [2] proposed the use of a tailored Unified Modeling Language (UML) or System Modeling Language (SysML) profile as the most powerful and extensible way to integrate an AUTOSAR method in company process flows.

An automotive tool-chain for AUTOSAR is also presented by Voget [25]. The work focuses on ARTOP, a common platform for innovations which provides common base functionality for development of AUTOSAR compliant tools. Unfortunately, the Eclipse-based ARTOP platform serves only as a common base for AUTOSAR tool development and is not a ready-to-use tool-solution. Moreover, ARTOP also requires time-consuming initial training before a tool can be developed.

The approach of bridging the gap between model-based system engineering and software engineering models based on EAST-ADL2 architecture description language and a complementary AUTOSAR representation is also very common in the automotive software development domain [5,14,23]. EAST-ADL represents an architecture description language using AUTOSAR elements to

represent the software implementation layer of embedded systems [1]. More recently the MAENAD Project[1] is also focusing on this approach.

Pagel et al. [16] mention the benefit of generating XML schema files directly from a platform-independent model (PIM) for data exchange via different tools. Performing extra transformation steps would only add potential sources for error and ambiguous mappings could result in unwanted side-effects.

Kawahara et al. [10] propose an extension of SysML which enables description of continuous time behavior. Their tool integration is based on Eclipse and couples SysML and Matlab/Simulink® via API .

Tool support for automotive engineering development is still organized as a patchwork of heterogeneous tools and formalisms [1]. On the one hand, general-purpose modeling languages (such as UML or SysML) provide modeling power suitable to capture system wide constraints and behavior, but lack in synthesizability. On the other hand, special-purpose modeling languages (such as C, Assembler, Matlab, Simulink, ASCET) are optimized for fine granular design and being less efficient in high-level design.

2.1 The Underlying Framework of the Proposed Approach

This section gives a brief overview of the underlying framework and related preliminary work which supports the proposed approach. The basic concept behind this framework is to have a consistent information repository as a central source of information, to store all information of all involved engineering disciplines of embedded automotive system development in a structured way. The concept focuses on allowing different engineers to do their job in their specific manner, but providing traces and dependency analysis of features concerning the overall system, e.g. safety, security, or dependability. Furthermore, the proposed approach is intended as an affordable, versatile, and tool independent method. This makes the method especially attractive for limited resources of small and micro-sized companies or small projects. Especially such projects or start-up companies often struggle with setting up their development processes or achieving adequate quality with limited resources (such as time or manpower). Therefore this approach stir out of common AUTOSAR based approaches and force a direct model transformation from SysML representation to Matlab/Simulink. The reason to make the decision of not fostering an AUTOSAR approach is based on one hand on focusing not only on AUTOSAR but rather generally on Matlab/Simulink based automotive software development. On the other hand, experiences we made with our previous approach [12] confirm the problem mentioned by Rodriguez et al. [20]. Not all tools fully support the whole AUTOSAR standard, because of its complexity, which leads to several mutual incompatibilities and interoperability problems.

The approach presented in this work and the work of Mader et al. [14] are based on similar concepts, but in contrast to their work, our technique supports automatic generation of whole software architectures, interface definition, timing

[1] http://maenad.eu/.

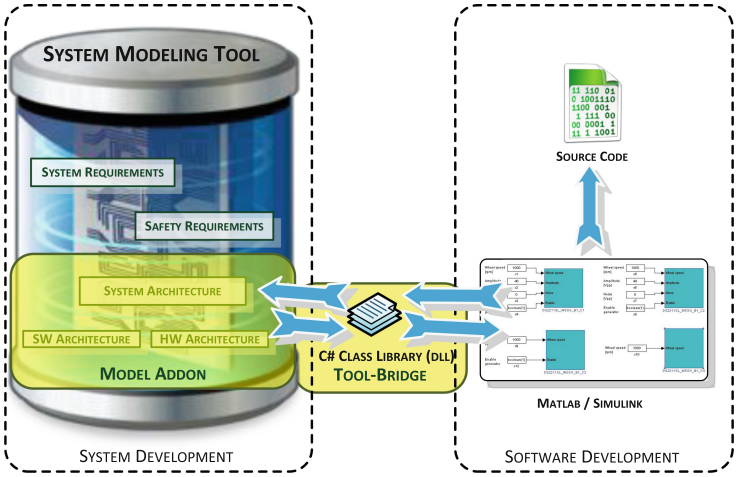


Fig. 1. Portrayal of the bridging approach transferring system development artefacts to SW development phase

setting, and auto-routing of signals in addition to their automatic generation of a software model. Figure 1 shows an overview of this approach and the embedded bridging of abstract system development and concrete software development models. For a more detailed overview of orchestration of the whole tool-chain see [13].

3 Model-Transformation Bridge

As mentioned in the previous section, the fundamental concept behind this framework is to have a consistent information repository as central source of information, to store all information of all involved engineering disciplines of embedded automotive system development in a structured way. This affordable, versatile, and tool independent approach forces a direct model transformation from SysML representation to Matlab/Simulink and is especially attractive for software development without full orchestration of AUTOSAR toolchain or non-AUTOSAR based development.

The contribution proposed in this work is part of the framework presented in [13] towards software development in the automotive context. More specifically, our contribution consists of the following parts:

- *UML Software Modeling Framework*: Enhancement of an UML profile for the definition of software development artifacts, more precisely, for the definition of the components interfaces and SW architecture composition. Required for consistent SW system description, see Fig. 1 – model addon.
- *SW Architecture Exporter*: Exporter to generate the designed SW architecture in the third party tool Matlab/Simulink for further detailed development, see Fig. 1 – tool bridge.

– *SW Architecture Importer*: Importer to integrate refined SW architecture and interfaces from the software development tool (e.g., as a result of round-trip engineering), see Fig. 1 – tool bridge.

This proposed approach closes the gap, also mentioned by Giese et al. [7], Holtmann et al. [8], and Sandmann and Seibt [21], between system-level development at abstract UML-like representations and software-level development modeling tools (e.g. Matlab/Simulink or Targetlink®). The bridging supports consistency of information transfer between system engineering tools and software engineering tools and minimizes redundant manual information exchange between these tools. This contributes to simplify seamless safety argumentation according to ISO 26262 [9] for the developed system. Benefits of this development approach are highly noticeable in terms of more efficient re-engineering cycles, and easy reuse of development artifacts with changing dependencies. As can be seen in Fig. 1, the lack of supporting tools for information transfer between system development tools and software development tools can be dispelled by our approach. The implementation of the bridge, based on versatile C# class libraries (dll) and Matlab COM Automation Server, ensures tool independence of the general-purpose UML modeling tool (such as Enterprise Architect or Artisan Studio) and version independence of Matlab/Simulink through API command implementation. This makes the method especially attractive for projects and companies with limited resources (such as manpower or finances). Especially small projects or start-up companies often struggle with setting up their development processes to achieve adequate quality.

3.1 UML Software Modeling Framework

The first part of the approach is the development of a specific UML modeling framework enabling software architecture design in AUTOSAR like representation within a state-of-the-art system development tool (in this case Enterprise Architect). This EA profile makes the UML representation more manageable for the needs of the design of an automotive software architecture by taking advantage of an AUTOSAR aligned abstraction layer. Furthermore, the profile enables an explicit definition of components, component interfaces, and connections between interfaces. This provides the possibility to define software architecture and ensures proper definition of the communication between the architecture artifacts, including interface specifications (e.g. upper limits, initial values, formulas). In addition this profile ensures the versatilely to also enable AUTOSAR aligned development as proposed in [12].

Hence, the SW architecture representation within EA can be linked to system development artifacts and traces to requirements can be easily established. This further benefits in terms of constraints checking, traceability of development decisions (e.g. for safety case generation), and reuse. Figure 2 shows an example of software architecture artifacts and interface information represented in Enterprise Architect.

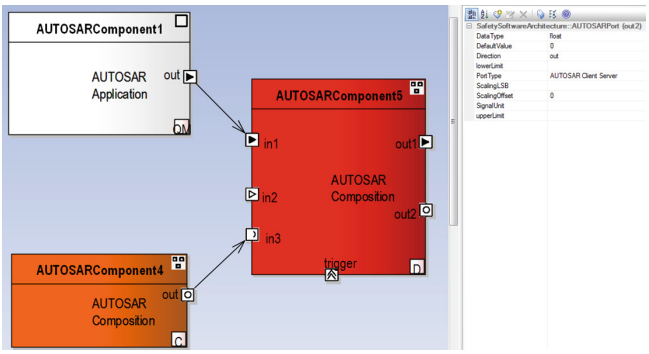


Fig. 2. Snapshot of the SW architecture representation within the system development tool and representation of the interface information

3.2 SW Architecture Exporter

The second part of the approach is an exporter which is able to export the software architecture, component containers, and their interconnections designed in SysML to the software development tool Matlab/Simulink. The implementation of the exporter is based on Matlab COM Automation Server and generates models through API command implementation. This ensures tool version-independence of the presented approach. Per user input the software architecture representation to be transferred is selected and a background task generates a corresponding Matlab/Simulink model. Listing 1.1 shows some excerpts of the automatically generated Matlab API commands. As can be seen in this listing, each model artifact, parameter, and connection is transferred to Matlab/Simulink, blocks are arranged and sized in correct manner and also unique links to the EA representation and assigned safety-criticality of the artifact (Listing 1.1 line 3 and 8) are established.

Listing 1.1. Excerpts of Matlab API commands

```
addpath(genpath('C:\EGasSystem'))
add_block('Simulink/Ports & Subsystems/Model','EGasSystem/EGasCtrl')
set_param('EGasSystem/EGasCtrl','ModelNameDialog','EGasCtrl',\
    ...'Description','EA_ObjectID@1969;ASIL@QM')
set_param('EGasSystem/EGasCtrl','Position',[250 50 550 250])

:
add_block('Simulink/Ports & Subsystems/In1','EGasSystem/APedl2')
set_param('EGasSystem/APedl2','Position',[50 200 80 215])
set_param('EGasSystem/APedl2','Outmin','0','Outmax','5',\
    ...'OutDataTypeStr','single','Description','EA_ObjectID@1966;\
    ...ASIL@B');

:
add_line('EGasSystem','APedl1/1','EGasMonr/1','AUTOROUTING','ON')

:
save_system('EGasSystem')
```

```
close_system('EGasSystem')
cd ..
cd C:\EGasSystem
```

If dSpace[2] tools are used for the subsequent C code generation instead of the Simulink Coder™, the software architecture can be exported into a TargetLink model optionally. In this case, the Matlab API command generator simply uses the *TargetLink common blockset* for creating the software framework model.

Furthermore, the signals from the software architecture design are analysed and transferred to the Simulink/TargetLink data dictionary by the exporter. This guarantees a consistent handling of the defined component interfaces as well as the connections between the interfaces throughout the development.

3.3 SW Architecture Importer

The last part of the approach is the import functionality add-on for the system development tool. This functionality, in combination with the export function, enables bidirectional update of software architecture representation in the system development tool and the software modules in Matlab/Simulink. The importer identifies the unique links to the EA representation (shown in Listing 1.1 line 3 and 8) and thereby differentiates new and modified model artifacts.

On the one hand, this ensures consistency between system development artifacts and changes done in the software development tool. On the other hand, the import functionality enables reuse of available software modules, guarantees consistency of information across tool boundaries, and shares information more precisely and less ambiguously.

Triggered via user input, a user interface within the system development tool (shown in Fig. 3) depicts modifications between the two representations and enables selective update of the UML based SW representation. As can be seen in Fig. 3, also a highlighting of the type of change (see Table 1) is provided.

Table 1. SW architecture importer indicators of type of change

Indicator	Type of change
A	Model artifact added
AC	Interface connection added
D	Model artifact deleted
DC	Interface connection deleted
U	Model artifact updated
UC	Interface connection updated

[2] http://www.dspace.com/.

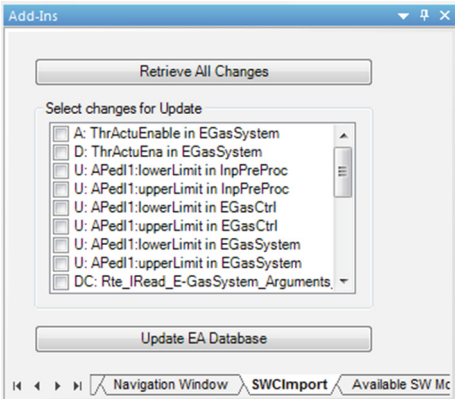


Fig. 3. SW architecture importer user interface

4 Application of the Proposed Approach

This section demonstrates the benefits of the introduced approach for the development of automotive embedded systems. To provide a comparison of the improvements of our approach, we use the 3 layer monitoring concept [26] as evaluation use-case. This elementary use-case is well-known in the automotive domain and because of this reason representative. This use-case is an illustrative material, reduced for internal training purpose of both, students and engineers. Therefore, the disclosed and commercially non-sensitive use-case is not intended to be exhaustive or representing leading-edge technology. An overview of the use-case is given in Table 2.

Table 2. Overview of the evaluation use-case SW architecture

Object type	Element-count	Configurable attributes per element
SW modules	7	3
SW interfaces	34	10
Connections	30	0

The definition of the software architecture is usually done by a software architect within the software development tool (Matlab/Simulink). With our approach, this work package is included in the system development tool (depicted in Fig. 4). This does not hamper the work of the software system architect but enables constraint checking features and helps to improve system maturity in terms of consistency, completeness, and correctness of the development artifacts. Beside this, the change offers a significant benefit for development of safety-critical software in terms of traceability, replicability of design decisions, visualizes dependencies unambiguously, and puts visual emphasis on view-dependent

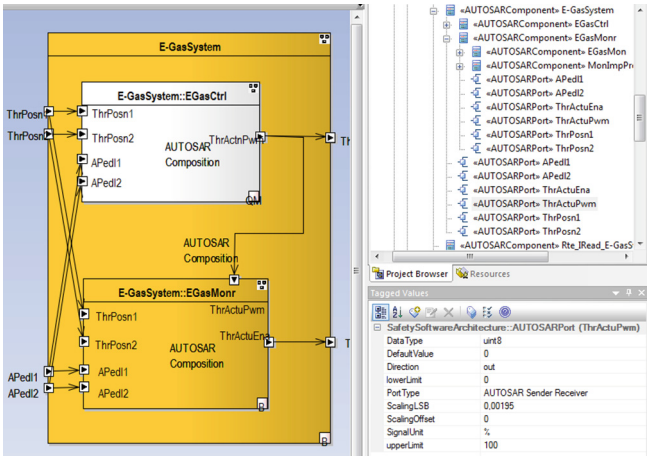


Fig. 4. Top-level representation of demonstration use-case in enterprise architect

constraints (such as graphical safety-criticality highlighting of SW modules in Fig. 4).

The presented use-case amounts to a total count of 41 model artifacts with 361 configuration parameters and 30 relations between the elements. This small example already indicates that relations between the model elements and number of model elements become confusing. Therefore, manual transformation of the information represented within the models is cumbersome and error-prone and would inherit lots of additional work to ensure consistency of both models in terms of safety-critical software development.

With our approach these information and model artifacts are checked for consistency constraints (such as point-to-point consistency of interface configurations) before automatically transferred via 212 lines of auto-generated Matlab API code. This auto-generation of Matlab API code provides evidence and ensures completeness of the model transformation. Furthermore, the SW import functionality enables round-trip engineering and bi-directional updates of both models and therefore supports evidence for consistency of both models.

According to the presented bridge approach in Sect. 3, the first step during the transformation is the decomposition of the software architectural design. Each software subsystem (like the AUTOSAR Composition *EGasCtrl* in Fig. 4) is analysed and the comprised software modules (e.g. *EGasCtrl::InpPreProc in Fig.* 5) are extracted. An essential information at the software module architecture is the trigger definition, representing the later task timing property of the module at the integrated system.

With the gathered architectural, timing, and interface information the Simulink/TargetLink model is generated by the previously described utilization of the Matlab COM Automation Server. During this process, a new Simulink root model is created and for each trigger type, which appears at the architectural design,

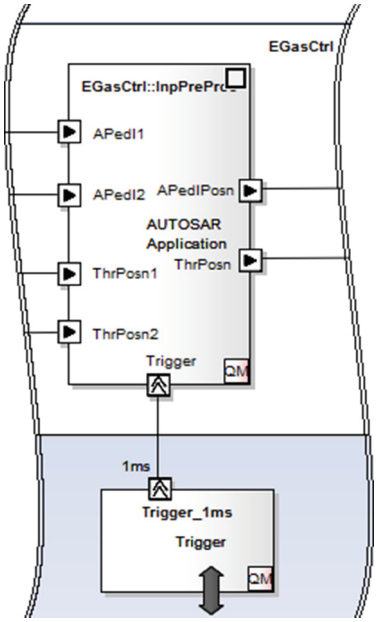


Fig. 5. Software module representation at the demonstrated use-case

an own subsystem is placed at the models top level. Afterwards, each software subsystem is transferred to the appropriate timing subsystem or even split up into multiple tasks if necessary. E.g. the subsystem *E-GasSystem::EGasCtrl* contains software modules with different timing attributes. Therefore, the *EGasCtrl*-Subsystem is available at multiple timing subsystems at the Simulink/TargetLink model. To facilitate a multi developer scenario, a separate model file is created for each software subsystem and linked as a reference model at the root model. To complete the Simulink/TargetLink model subsystem generation, all software modules are transferred into their appropriated software subsystem.

The transformation process described so far, generates a software framework out of the *Autosar Composition* and *Autosar Application* blocks at the design. To provide a complete model framework, which serves as a basis for the subsequent software unit development, the interfaces as well as their connections are transferred to Simulink/TargetLink in the next step. For an efficient and dependable handling of the signals, a *Data Dictionary* and the related tool *Data Dictionary Manager* is used. Again, to facilitate a multi developer scenario, a data dictionary file is created for each software subsystem and included at a data dictionary root file. Every signal from the architectural design is stored at the appropriate data dictionary, including all available attributes like value limits, scaling, etc. Furthermore, the exporter algorithm simply link this entry wherever the signal occurs in the Simulink/TargetLink model (see Fig. 6).

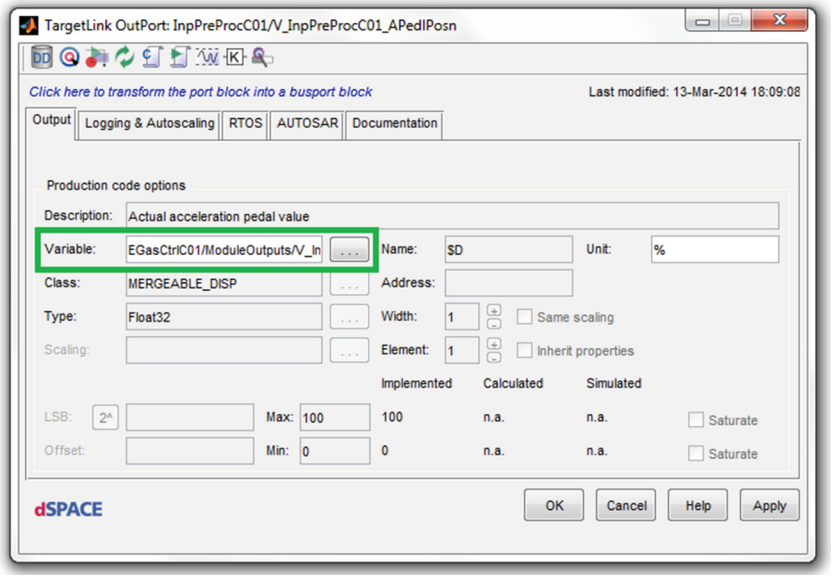


Fig. 6. Link to the signal at the data dictionary, set by the exporter

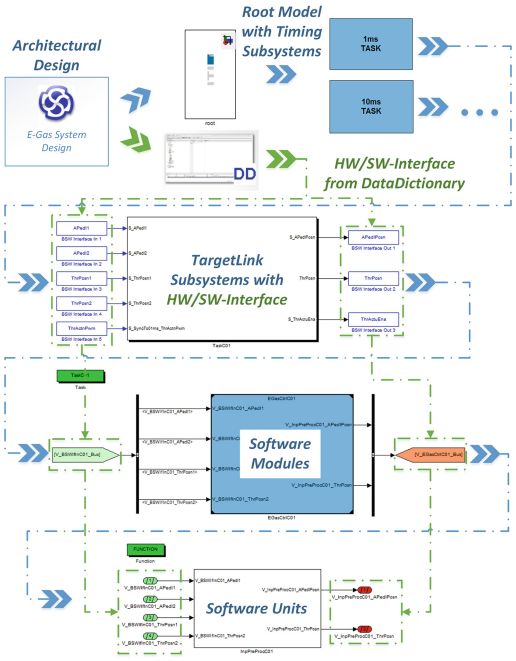


Fig. 7. Architectural design to Simulink/TargetLink model transformation workflow

The process workflow *Architectural Design to Simulink/TargetLink Model Transformation* is shown in Fig. 7, with the blue-coloured subsystem creation path, and the green-coloured signal creation path. Although, the chosen use-case example is not too complex, presenting all generated artifacts would go beyond the scope of this contribution. Therefore, the illustration showcases the creation of the Simulink/TargetLink *root model*, the software subsystem *EGasCtrl* including the signals from and to the basis software, and the software module *InpPreProc*, which was already presented in Fig. 5.

If the Simulink/TargetLink model already exists when the *Software Architecture Exporter* is triggered, the *Software Architecture Importer* is started automatically in the background to check the consistency between the architectural design and the software model. If all artifacts at the model are available at the design, and new items exist at the design, the software model is updated by the exporter analogue to the procedure described above for completely new Simulink/TargetLink models. If there are new or deleted artifacts at the software model, a notification is displayed and the user is prompted to determine the further procedure, like shown in Fig. 3, Sect. 3.

5 Conclusion

Dependable system development is an emerging trend in automotive industry, aiming to provide a convincing argumentation that the system under development has achieved a certain level of maturity. Without an adequate tool chain, which enables a smooth transition between the different levels along the system development, it is hard to obtain this demanded maturity.

Especially creating a software model from the architectural design manually is exhausting and error-prone. The risk to e.g. connect signals incorrect, set wrong attributes or simply overlook a changed parameter is very high.

This paper presented an efficient approach to avoid the risk of introducing errors while developing the software according to the architectural design, by creating the software model framework fully automated. Furthermore, the concept facilitates bidirectional traceability as well as consistency. These properties are elemental key factors for a high quality development and postulated by the widespread quasi-standard *Automotive SPICE*. Additionally, the shown techniques facilitate round-trip engineering by the presented import/export functionality regarding the models on different development levels and tools.

In terms of safety-critical development and reuse the presented approach features are crucial to transfer information between separated tools and link supporting safety-relevant information. Moreover, the approach eliminates the need of manual information rework without adequate tool support, ensuring reproducibility, and traceability argumentation.

The application of the presented approach has been demonstrated utilizing a simplified version of the well-known E-Gas concept, which is intended to be used for training purpose of students and engineers and not for representing an exhaustive or commercial sensitive project.

References

1. Blom, H., Loenn, H., Hagl, F., Papadopoulos, Y., Reiser, M.-O., Sjoestedt, C.-J., Chen, D., Kolagari., R.: EAST-ADL - an architecture description language for automotive software-intensive systems. White Paper 2.1.12 (2013)
2. Boldt, R.: Modeling AUTOSAR systems with a UML/SysML profile. Technical report, IBM Software Group (2009)
3. Broy, M., Feilkas, M., Herrmannsdoerfer, M., Merenda, S., Ratiu, D.: Seamless model-based development: from isolated tool to integrated model engineering environments, IEEE Magazin (2008)
4. Buchmann, T., Westfechtel, B.: Towards incremental round-trip engineering using model transformations. In: 2013 39th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), pp. 130–133, Sept 2013
5. Chen, D.J., Johansson, R., Lönn, H., Papadopoulos, Y., Sandberg, A., Törner, F., Törngren, M.: Modelling support for design of safety-critical automotive embedded systems. In: Harrison, M.D., Sujan, M.-A. (eds.) SAFECOMP 2008. LNCS, vol. 5219, pp. 72–85. Springer, Heidelberg (2008)
6. Ebert, C., Jones, C.: Embedded software: facts, figures, and future. IEEE Comput. Soc. **0018–9162**(09), 42–52 (2009)
7. Giese, H., Hildebrandt, S., Neumann, S.: Model synchronization at work: keeping SysML and AUTOSAR models consistent. In: Engels, G., Lewerentz, C., Schäfer, W., Schürr, A., Westfechtel, B. (eds.) Nagl Festschrift. LNCS, vol. 5765, pp. 555–579. Springer, Heidelberg (2010)
8. Holtmann, J., Meyer, J., Meyer, M.: A seamless model-based development process for automotive systems (2011)
9. ISO - International Organization for Standardization. ISO 26262 Road vehicles functional safety, Part 1–10 (2011)
10. Kawahara, R., Dotan, D., Sakairi, T., Ono, K., Kirshin, A., Nakamura, H., Hirose, S., Ishikawa, H.: Verification of embedded system's specification using collaborative simulation of SysML and Simulink models. In: Proceedings of Second International Conference on Model Based Systems Engineering, pp. 21–28, March 2009
11. Lau, K.-K., Tepan, P., Tran, C., Saudrais, S., Tchakaloff, B.: A holistic (Component-based) approach to AUTOSAR designs. In: 2013 39th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), pp. 203–207, Sept 2013
12. Macher, G., Armengaud, E., Kreiner, C.: Automated generation of AUTOSAR description file for safety-critical software architectures. In: 12. Workshop Automotive Software Engineering (ASE), Lecture Notes in Informatics (2014)
13. Macher, G., Armengaud, E., Kreiner, C.: Bridging automotive systems, safety and software engineering by a seamless tool chain. In: 7th European Congress Embedded Real Time Software and Systems Proceedings, pp. 256–263 (2014)
14. Mader, R., Griessnig, G., Eric, A., Andrea, L., Christian, K., Bourrouilh, Q., Steger, C., Weiss, R.: A bridge from system to software development for safety-critical automotive embedded systems. In: 38th Euromicro Conference on Software Engineering and Advanced Applications, pp. 75–79 (2012)
15. Meyer, J.: Eine durchgaengige modellbasierte Entwicklungsmethodik fuer die automobile Steuergeraeteentwicklung unter Einbeziehung des AUTOSAR Standards. Ph.D thesis, Universitaet Paderborn, Fakultaet fuer Elektrotechnik, Informatik und Mathematik, July 2014

16. Pagel, M., Brörkens, M.: Definition and generation of data exchange formats in AUTOSAR, process independent model. In: Rensink, A., Warmer, J. (eds.) ECMDA-FA 2006. LNCS, vol. 4066, pp. 52–65. Springer, Heidelberg (2006)
17. Petrissans, A., Krawczyk, S., Veronesi, L., Cattaneo, G., Feeney, N., Meunier, C.: Design of future embedded systems toward system of systems - trends and challenges. European Commission, May 2012
18. Pretschner, A., Broy, M., Kruger, I.H., Stauner, T.: Software engineering for automotive systems: a roadmap. In: 2007 Future of Software Engineering, FOSE 2007, Washington, DC, USA, pp. 55–71, IEEE Computer Society (2007)
19. Quadri, I.R., Sadovykh, A.: MADES: a SysML/MARTE high level methodology for real-time and embedded systems (2011)
20. Rodriguez-Priego, E., Garcia-Izquierdo, F., Rubio, A.: Modeling issues: a survival guide for a non-expert modeler. Models **2010**(2), 361–375 (2010)
21. Sandmann, G., Seibt., M.: AUTOSAR-compliant development workflows: from architecture to implementation - tool interoperability for round-trip engineering and verification and validation. In: SAE World Congress and Exhibition 2012, (SAE 2012–01-0962) (2012)
22. Scuro, G.: Automotive industry: innovation driven by electronics (2012). http://embedded-computing.com/articles/automotive-industry-innovation-driven-electronics/
23. Sjoestedt, C.-J., Shi, J., Toerngren, M., Servat, D., Chen, D., Ahlsten, V., Loenn, H.: Mapping simulink to UML in the design of embedded systems: investigating scenarios and structural and behavioral mapping. In: OMER 4 Post Workshop Proceedings, April 2008
24. Thyssen, J., Ratiu, D., Schwitzer, W., Harhurin, E., Feilkas, M., München, T.U., Thaden, E.: A system for seamless abstraction layers for model-based development of embedded software. In: Software Engineering Workshops, pp. 137–148 (2010)
25. Voget, S.: SAFE RTP: an open source reference tool platform for the safety modeling and analysis. In: Embedded Real Time Software and Systems Conference Proceedings (2014)
26. Zurawka, T., Schaeuffele, J.: Method for checking the safety and reliability of a software-based electronic system, January 2007

Tejo: A Supervised Anomaly Detection Scheme for NewSQL Databases

Guthemberg Silvestre[1,2(✉)], Carla Sauvanaud[1,3], Mohamed Kaâniche[1,2], and Karama Kanoun[1,2]

[1] CNRS, LAAS, 7 Avenue du Colonel Roche, 31400 Toulouse, France
{gdasilva,csauvana,mohamed.kaaniche,karama.kanoun}@laas.fr
[2] Univ de Toulouse, LAAS, 31400 Toulouse, France
[3] Univ de Toulouse, INSA de Toulouse, LAAS, 31400 Toulouse, France

Abstract. The increasing availability of streams of data and the need of auto-tuning applications have made big data mainstream. NewSQL databases have become increasingly important to ensure fast data processing for the emerging stream processing platforms. While many architectural improvements have been made on NewSQL databases to handle fast data processing, anomalous events on the underlying, complex cloud environments may undermine their performance. In this paper, we present Tejo, a supervised anomaly detection scheme for NewSQL databases. Unlike general-purpose anomaly detection for the cloud, Tejo characterizes anomalies in NewSQL database clusters based on Service Level Objective (SLO) metrics. Our experiments with VoltDB, a prominent NewSQL database, shed some light on the impact of anomalies on these databases and highlight the key design choices to enhance anomaly detection.

1 Introduction

Big data has transformed the way we manage information. As an unprecedented volume of data has become available, there is an increasing demand for stream processing platforms to transform raw data into meaningful knowledge. These velocity-oriented platforms may rely on cloud databases to provide fast data management of continuous and contiguous flows of data with horizontal scalability. Therefore, cloud databases represent an important technology component for a broad range of data-driven domains, including social media, online advertisement, financial trading, security services, and policy-making process.

The architecture of row-store-based relational databases has evolved to meet the requirements of big data on the cloud [19], like elasticity, data partitioning, shared nothing, and especially high performance. The so-called NewSQL databases offer high-speed, scalable data processing in main-memory with consistency guarantees through ACID (atomicity, consistency, isolation, and durability) transactions.

To ensure fast data management, NewSQL databases rely on built-in, fault-tolerance mechanisms, like data partitioning, replication, redundant network

A. Fantechi and P. Patrizio (Eds.): SERENE 2015, LNCS 9274, pp. 114–127, 2015.
DOI: 10.1007/978-3-319-23129-7_9

topologies, load balancing, and failover. Although these mechanisms handle fail-stop failures successfully, many other cloud performance anomalies may remain unnoticed [20]. For instance, Do *et al.* [8] found that a single limping network interface can cause a three orders of magnitude execution slowdown in cloud databases. Therefore, we believe that the dependability of NewSQL databases might be improved by detecting these anomalies.

This paper proposes Tejo, a supervised anomaly detection scheme for NewSQL databases. We make three specific contributions. First, we introduce a scheme for analysing performance anomalies using fault injection tools and a supervised learning model. Second, we shed some light on the impact of performance anomalies in NewSQL databases. Third, we highlight the importance of selecting the proper features and statistical learning algorithm to enhance the anomaly detection efficiency on these databases.

In the next section, we lay out the recent trends in data stream processing and anomaly detection with statistical learning. Following this, in Sect. 3 we describe the design of Tejo, in particular its components and its two-phased functioning, namely learning and detection phase. In Sect. 4 we evaluate VoltDB, a prominent NewSQL database, using Tejo. In our experimental setup, VoltDB served two workloads, whose data was partitioned and replicated across a cluster of virtual machines (VMs). Finally, we discuss the related work in Sect. 5, and conclude in Sect. 6.

2 Background

2.1 Big Data Stream Processing and NewSQL Databases

To processing continuous streams of big data, we consider the emerging stream processing platforms [15], as depicted in Fig. 1. In these platforms, streams of data are processed by two complementary systems: the fast stream processing system and big archival engine. The former manages high-speed data streams to provide real-time analytics and data-driven decisioning, providing services like fraud heuristics, market segmentation, or optimal customer experience; while the later computes huge volumes of historical data for long-term data analytics, such as scientific results, seasonal predictions, and capacity planning. Big archival engines are built on data warehouse technologies like Hadoop and column-stores. In contrast, fast stream processing technologies are still emerging. Among these technologies are NewSQL databases like VoltDB [23] and

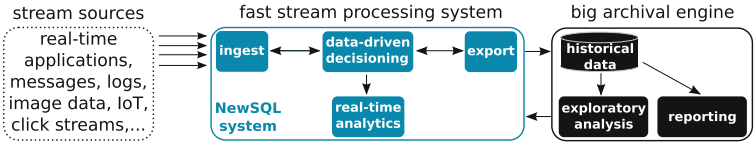


Fig. 1. The emerging stream processing platforms for big data.

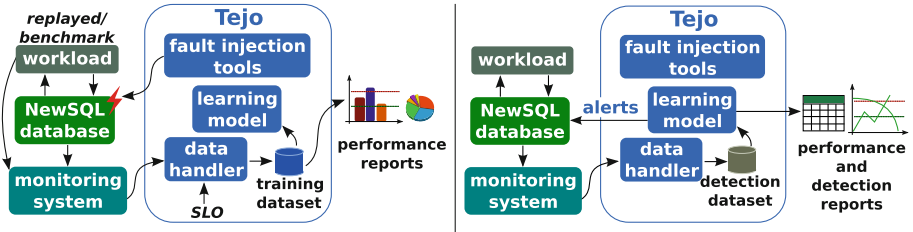
S-Store [4]. To support incremental, stateful ingest of data streams into a scalable system, NewSQL databases provide low-latency via in-memory distributed processing and a strong support for transaction management with ACID guarantees. However, as NewSQL databases are deployed on cloud infrastructures to scale to large clusters, cloud performance anomalies may undermine their capacity of fast stream processing.

2.2 Anomaly Detection Using Statistical Learning

Statistical learning has been a widely used technique to predict performance anomalies in large-scale distributed systems [5]. It makes prediction by processing feature vector $\mathbf{x}$ with a fixed number of dimensions d ($\mathbf{x} \in \mathcal{X} \subset \mathbb{R}^d$) from the input space $\mathcal{X}$. There are two main methods: *supervised* and *unsupervised learning*.

The *supervised learning method* couples each input with a y, a label, from the output space $\mathcal{Y}$. To learn, we have N pairs $(\mathbf{x}, y)$ drawn *independent and identically distributed* (i.i.d.) from a fixed but unknown joint probability density function $Pr(X, Y)$. This method searches for a function $f : \mathcal{X} \rightarrow \mathbb{R}$ in a fixed function class $\mathcal{F}$ in the learning dataset. State-of-the-art algorithms, like *random forests* [2], aim to find $f^\star$ in $\mathcal{F}$ with the lowest empirical risk $f^\star \in \arg\min_{f \in \mathcal{F}} \mathtt{r}_{emp}(f)$, where $\mathtt{r}_{emp}(f) = \frac{1}{N} \sum_{i=1}^{N} I_{\{f(\mathbf{x}) \neq y_i\}}$ is computed over the training set, and $I_{\{.\}}$ is the indicator function which returns 1 if the predicate $\{.\}$ is true and 0 otherwise. Similarly, an *unsupervised learning method* relies on N unlabelled samples having probability density function $Pr(X)$. Unlike supervised learning, predictions provide insights into how the data is organized or clustered.

Most of the anomaly detection approaches for distributed systems are based on a general-purpose, unsupervised learning method [12–14]. However, prediction efficiency remains the main drawback of this method [16]. Results in our previous work [21] confirm that a supervised learning method overcomes an unsupervised one in cloud anomaly detection. In this work, we extend our supervised learning model as a component of Tejo to classify anomalous VMs in four different classes.



(a) Learning phase. (b) Detection phase.

Fig. 2. Tejo operates in two distinct phases: learning and detection phase.

3 Approach

Tejo comprises three components, namely a set of fault injection tools, a data handler, and a learning model. These components interoperate into two distinct phases: learning and detection phase. While the first phase allows us to evaluate the performance of a NewSQL database under anomalies, the second permits the detection of these anomalies. Figure 2 depicts the components and the two-phase functioning of Tejo.

3.1 The Components of Tejo

Fault Injection Tools. To provoke performance anomalies, this component emulates four categories of faulty events in VMs of a NewSQL database cluster.

Network faults. Communication issues are common in distributed systems. To analyse their impact, we inject three types of network faults, namely packet loss, network latency, and limping network. Packet loss and network latency emulates interconnection issues, such as network partition. Limping network reproduces anomalies previously observed by Do *et al.* [8], where the transmission rate of limping network interface is smaller than the manufacturer's specification.

Memory faults. As NewSQL databases fit the entire data to the main memory, they become more vulnerable to anomalies in memory availability. To provoke such anomalies, we make arbitrary amounts of main memory unavailable. As a result, the database instance is likely to perform more costly disk I/O operations. A typical example of this fault is a VM running out of memory due to a misconfiguration, memory leaking, overloading, or an unbalanced resource allocation.

Disk faults. Although most NewSQL databases manage data in main memory, disk-intensive processes may have an impact of its performance. This category of fault emulates an arbitrary number of jobs performing several disk operations, including writes, reads, and file syncs.

CPU faults. CPU is a key resource in a virtual machine. As unattended number of processes compete the database instance to CPU resources, they may undermine the performance of the database cluster. The CPU fault emulates an arbitrary number of jobs performing arithmetic operations to overload the VM cores.

The Data Handler. This component computes data from the monitoring system (i) to collect the performance counters of a NewSQL database and (ii) to provide data to characterize performance anomalies.

Collect the performance counters of a NewSQL database. The data handler samples monitoring data to collect the current state of the NewSQL cluster, as depicted in Fig. 2. To this end, it frequently communicates with the monitoring system to fetch raw monitoring data and to convert it into useful, aggregated

information. The content of the resulting aggregated information depends on the aim of each functioning phase, learning or detection, detailed below.

Providing data to characterize performance anomalies. After aggregating samples of monitoring data, the data handler organizes this data into *feature vectors*. These vectors represent the state of the VMs of the database cluster or the workload. The vectors are stored in datasets for performance analysis or anomaly detection.

The Learning Model. The learning model is at the heart of the Tejo scheme. The purpose of this model, the so-called predictive task, is to characterize the behaviour of VMs under performance anomalies. Given an i.i.d. sample $(\mathbf{x}, y)$, described in Subsection 2, we model our predictive task as a classification problem, whose inputs and outputs are defined as follows.

Inputs. We represent the input space $\mathbf{x}$ as a VM running a database instance. This input data corresponds to a feature vector computed by the data handler component. The size of the feature vector matters. In general, the higher the dimension of this vector, the higher the predictive efficiency is. However, an increase in the input dimension rises the computational cost of predictions.

Outputs. The supervision y associated to each input VM $\mathbf{x}$ is based on five possible classes, $\mathcal{Y} \in \{0, 1, 2, 3, 4\}$, whose labels are *normal*, *network-related anomaly*, *memory-related anomaly*, *disk-related anomaly*, and *CPU-related anomaly* respectively. Depending on the phase of Tejo (detailed below), these labels are assigned by either computing the training dataset or by a learning algorithm.

3.2 Two-Phase Functioning

Learning Phase. In this phase, Tejo learns the behaviour of the database cluster under anomalies and reports on its performance.

Requirements. As illustrated in Fig. 2a, Tejo relies on an already existing monitoring system to poll performance counters from both VMs and workload. To measure the cluster-wide performance counters, we assume that the workload can be replayed or run through a benchmark tool. We consider that Tejo's analyst specifies expected SLO metrics, such as average throughput and 99^{th} percentile latency. The analyst must also specify parameters of the fault campaign and running experiments, including intensity and duration of each fault, number of injections, and interval between consecutive fault injections.

Functioning. As the replayed/benchmark workload runs, the fault injection tool performs a fault injection campaign to emulate performance anomalies. Meanwhile, performance counters from both the workload and VMs running the database cluster are collected by the monitoring system and computed by the data handler component. As the data handler computes the monitoring data in feature vectors, it adds information about injected faults and labels. Labels correspond to output classes of learning model and are added with respect to SLO metrics.

The feature vectors are then stored in the training dataset. After running the workload and accomplishing the fault injection campaign, the learning model computes the feature vectors of VMs from the training dataset.

Reports. Besides providing data to learn the behaviour of anomalous VMs, analysts can observe the impact of anomalies in the throughput and 99^{th} percentile latency. They may evaluate which anomalies cause SLO violations, gaining more insight into the efficiency of existing fault-tolerance mechanisms.

Detection Phase. In this phase, Tejo reports on the efficiency of the learning model and performs anomaly detection in VMs at runtime.

Requirements. Similar to the training phase, Tejo relies on an existing monitoring platform to gather data for predictions. It requires that the learning model has already been trained as detailed in learning phase described above. We assume that SLO targets and the workload are the same as those of the learning phase.

Functioning. While the NewSQL database serves the workload, the data handler gathers the monitoring data and creates feature vectors of VMs in the detection dataset. As soon as a new feature vector is created, the learning model computes it to detect performance anomalies whenever they occur.

Reports. Tejo's learning model predicts labels of incoming feature vectors. Then alerts are generated about detected anomalies. These alerts may be handled by the database to trigger recovery procedures. Besides generating alerts, it reports on the efficiency of the learning model, including comparing different learning algorithms, ranking performance counters with regard to their importance, calculating the computational cost, and verifying model over-fitting or under-fitting.

4 Evaluation

We evaluate VoltDB, a NewSQL database, with Tejo. First, we describe our experimental setup. Second, we measure the impact of performance anomalies in a VoltDB cluster. Finally, we report on the predictive efficiency of these anomalies.

4.1 Experimental Setup

We performed our experiments on a private cloud consisting of two Dell PowerEdge R620 hosts. Each host has two-core Xeon E5-2660 at 2.2 GHz, 64 GB of memory, and two 130 GB SATA disks. Hosts are connected by Gigabit Ethernet. We chose VMware as the virtualization technology and ESXi 5.1.0 as hypervisor. Figure 3 depicts our private cloud, highlighting the consolidation of VMs. Each VM of the NewSQL database cluster has 4 GB of memory, 4 CPU cores, a disk of 16 GB, and is connected to a 100 Mbps virtual network.

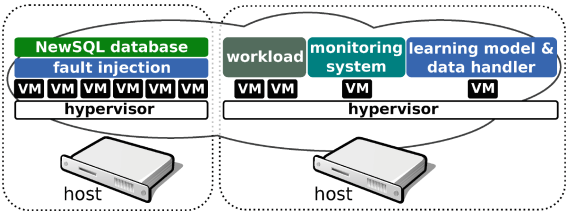


Fig. 3. Experimental setup.

Components of Tejo and Monitoring System. As fault injection tools, we chose Dummynet (v3.0) [3] for network faults and stress-ng (v0.01.30)[1] for disk, memory, and CPU faults. These tools provide a flexible, easy-to-reproduce way to inject arbitrary fault intensities. Table 1 lists the parameters of our fault injection campaign. The data handler component was implemented as a collection of python/shell scripts along with PostgreSQL database for datasets. We implemented our learning model using the Scikit-learn library [17], from which we evaluated three learning algorithms: random forests [2], gradient boosting [10], and SVM [7]. We used Ganglia as monitoring system. Our setup required additional Ganglia plug-ins for collecting performance counters of the workload and VoltDB. Every 15 seconds, we collected 147 performance metrics of each VM, and the average throughput and the 99^{th} percentile latency from the served workload.

Table 1. The key parameters of Tejo for our fault injection campaign.

Fault		Intensity ranges			Unit
		Light	Medium	Heavy	
Network	Pkt loss	1.6–3.2	4–5.6	6.4–8	%
	Latency	8–20	26–38	44–56	ms
	Limping	85-65	56-38	29-11	Mbps
Memory		73–79	82–88	91–97	%
Disk		10–20	25–35	40–50	writers
CPU		19–39	49–69	79–99	%

NewSQL Database and Workloads. We evaluated VoltDB (v4.x) as NewSQL database. We set the number of partitions VoltDB to 18 across a cluster of six VMs with failover mechanisms enabled. We varied the replication degree k from two to zero (i.e., replication disabled). We evaluated VoltDB with two workloads, the popular TPC-C benchmark for OLTP[2], and Voter[3], a workload

[1] stress-ng. http://kernel.ubuntu.com/~cking/stress-ng/.
[2] TPC-C benchmark (v5.10). http://www.tpc.org/tpcc/.
[3] Voter. https://github.com/VoltDB/voltdb/tree/master/examples/voter.

derived from leaderboard maintenance application for Japanese version of the "American Idol".

4.2 Evaluating Performance Anomalies in VoltDB

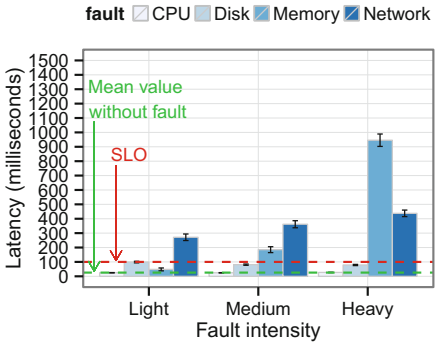
We run Tejo in learning mode (Subsect. 3.2) to evaluate the impact of faults in VoltDB. We selected a dataset containing 200,000 samples, including performance counters of VMs and the workload. Data was evenly collected across the two evaluated workloads. Figure 4 shows the impact of faults on the performance of VoltDB with a replication degree k=2. For each workload, they show the resulting performance anomalies on the average throughput and 99^{th} percentile latency, including mean values without faults, the expected SLO metrics, and 95 % confidence interval for performance metrics under fault injection.

Overall, the impact of increasing levels of faults was higher on the 99^{th} percentile latency than the average throughput. For instance, Fig. 4a shows that the 99^{th} percentile latency of VoltDB serving Voter workload soars under faults, especially for network and memory faults. Although the mean of the 99^{th} percentile latency without fault was 25 ms, it reaches 945 ms under memory faults. Similar results were found as VoltDB served TPC-C workload. However, we noticed that TPC-C has a greater performance degradation under memory faults (Fig. 4c). The reason for that is the main memory usage of each workload. While Voter uses 25 % of main memory from each VM, TPC-C utilises almost 50 %. Consequently, TPC-C is more sensitive to memory faults than Voter. Disk faults had a limited impact of the performance of VoltDB, slightly higher on TPC-C than Voter due to a greater need to synchronize data from the main memory to disk (Fig. 4c). Surprisingly, CPU faults had no impact on the performance of both workloads, even under heavy fault intensity (i.e., 99 % of CPU usage).

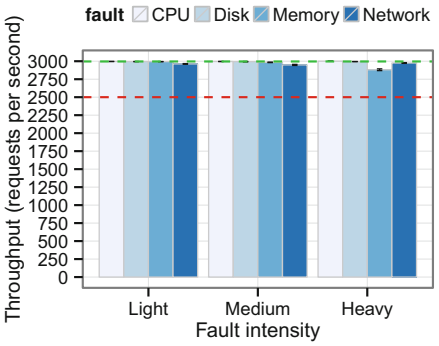
To shed some light on the capacity of data replication to mitigate the impact of performance anomalies, we varied the replication degree k of VoltDB from two to zero (i.e., replication disabled). Figure 5 shows a summary of the results of the impact of faults with medium intensity on VoltDB. In general, our results suggest that higher the replication degree the worse is the performance. The reason is that NewSQL databases as VoltDB strive to provide ACID properties both for concurrent transactions and for replicas. The impact of a fault on a single node spreads across the replicas on the cluster more easily, worsening its performance.

4.3 Predictive Efficiency Analysis

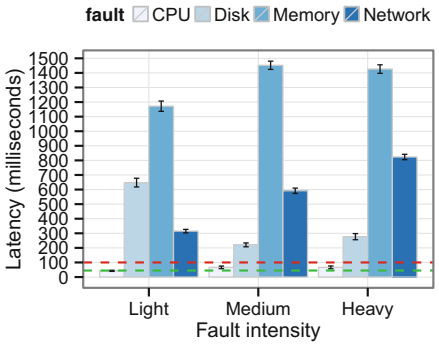
We evaluated the predictive efficiency of learning model of Tejo with three learning algorithms, random forests, gradient boosting, and SVM. To this end, we used data derived from the dataset of Subsect. 4.2. The derived data was computed by the Tejo's data handler, as described in Subsect. 3.2. Each new sample had 147 features ($d = 147$, as discussed in Subsect. 2.2) and a label corresponding to a class of Tejo's learning model (see Subsect. 3.1). Recall that anomaly-related labels are only assigned to samples that violated the SLO. The resulting dataset



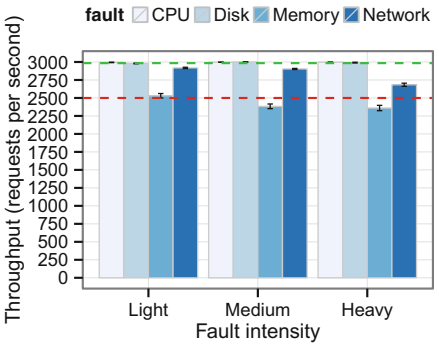
(a) 99^{th} percentile latency w/ Voter.



(b) Average throughput w/ Voter.

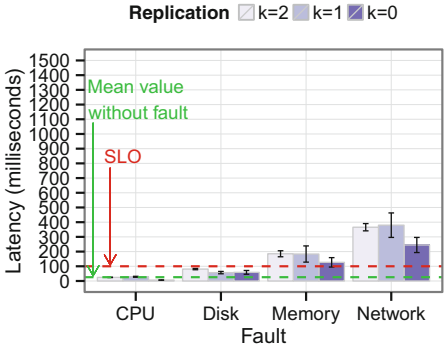


(c) 99^{th} percentile latency w/ TPC-C.

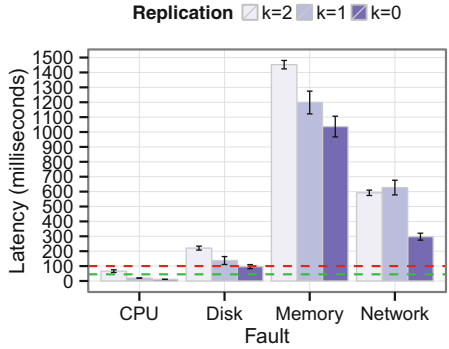


(d) Average throughput w/ TPC-C.

Fig. 4. Performance anomalies in VoltDB as serving Voter and TPC-C, with a replication degree k=2.



(a) Voter workload.



(b) TPC-C workload.

Fig. 5. Medium fault impact on VoltDB with different k values.

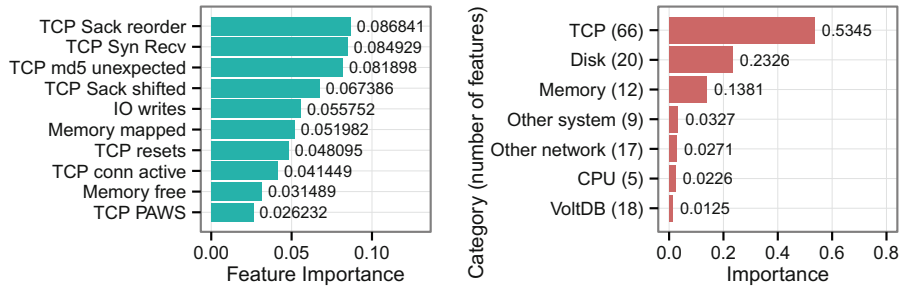
contained 10,000 samples for each evaluated workload, including 5,000 of samples representing anomalous events in VMs. To validate the learning model properly, this dataset was split in two uneven parts: three-fifths of data for training the model and two-fifths for testing its predictive efficiency. We used two well-known measures to evaluate the learning model efficiency, precision and F1-score. We also computed the overhead of predictions with each learning algorithm.

Table 2 summarizes our results. Regardless the learning algorithm, the learning model of Tejo was able to detect 96 % of anomalies properly. It performed better with random forests algorithm, whose overall score was 0.99 (up to 1) for both precision and F1-score measures. Random forests also provided the lowest overhead for anomaly detection, requiring less than 30 microseconds for a prediction. The SVM algorithm had the worst predictive performance, particularly to detect memory-related anomalies. SVM also incurred the highest overhead for anomaly detection with our model, performing two orders of magnitude slower. According to Friedman [10], this happens because SVM shares the disadvantages of ordinary kernel methods, such as poor computational scalability and inability to deal with irrelevant features. In contrast, boosting methods, like random forests and gradient boosting, overcome these issues by using a linear combination of (many) trees.

Table 2. Anomaly detection performance with different learning algorithms.

Algorithm	Class	Workload					
		Voter			TPC-C		
		Precision	F1-score	Overhead	Precision	F1-score	Overhead
Random forests	Normal	0.99	0.99	23 μs	0.98	0.99	26 μs
	Network	0.98	0.98		0.99	0.98	
	Memory	0.99	0.98		0.98	0.99	
	Disk	0.99	0.98		1.00	1.00	
Gradient boosting	Normal	0.99	0.99	30 μs	0.96	0.98	33 μs
	Network	0.99	0.99		0.99	0.96	
	Memory	0.99	0.99		0.98	0.99	
	Disk	1.00	1.00		1.00	1.00	
SVM	Normal	0.98	0.97	4294 μs	0.98	0.98	5441 μs
	Network	0.97	0.96		0.99	0.97	
	Memory	0.85	0.91		0.87	0.93	
	Disk	1.00	0.97		1.00	1.00	

In addition to the predictive efficiency evaluation, Tejo allows us to analyse the importance of features using boosting methods. Figure 6 plots the importance of features of the Tejo's learning model, where the sum of all features importances is equal to one. Figure 6a shows the 10 most-important features for anomaly detection in VoltDB serving Voter, seven out of 10 corresponding to performance counters of TCP layer of VMs. This suggests that the peer-to-peer communication pattern among the VoltDB cluster is key for anomaly detection.



(a) The 10 most-important features. (b) Importance of features categories.

Fig. 6. Analysis of the importance of features for anomaly detection.

To provide insights into all 147 features, we organized them into seven distinct categories and measured their grouped importance, as depicted in Fig. 6b. Indeed, it confirms that features from TCP performance counters form the main category, accounting for more than half the total of importance (0.5345). Surprisingly, the category of VoltDB features had the lowest importance for the anomaly detection task. This suggests that the contribution of database-specific features is negligible, therefore our learning model is likely to have similar predictive performance with different NewSQL databases. Results for TPC-C workload showed a similar trend.

5 Related Work

Fault Tolerance in Distributed Databases. Distributed databases use replication and advanced request scheduling to improve data availability. Bayou [18] is data storage that relies on replication to ensure data availability against fail-stop failures, but they are not able to deal with performance anomalies. Skute [1] provides an adaptive replication scheme that mitigates the impact of performance anomalies. However, it does not provide mechanisms to ensure high data availability, such as high throughput and bounded latency. Emerging cloud databases, like VoltDB and MongoDB, offer high data availability using enhanced main memory data structures [22]. But, our findings of this and previous study [21] show that performance anomalies on the cloud, including malfunctioning network cards, disk and main memory, can undermine the performance of cloud databases.

Cake [25] offers a scheduling scheme to enforce high-level data availability requirements for end users. However, Cake was not designed to identify faulty VMs. Eriksson *et al.* [9] provide a routing framework that helps cloud operators to mitigate the impact of network failures. We believe that our work is complementary to theirs. Alerts from Tejo about anomalies in network, memory, disk and CPU of VMs, can contribute to enhance the efficiency of such scheduling mechanisms.

Anomaly Detection with Statistical Learning. Anomaly detection is commonly implemented based on an *unsupervised learning method*. Gujrati *et al.* [13] provide prediction models based on event logs of supercomputers to detect platform-wide anomalies, whereas we are interested in detecting anomalous VMs based on monitoring data. Chen *et al.* [6] propose an anomaly detection approach for large-scale systems that improves the prediction efficiency of an entropy-based information theory technique by performing a principal component analysis (PCA) of system inputs. However, this introduces computational overhead that undermines its scalability and causes a slowdown in anomaly predictions. While we focus on detecting performance anomalies in NewSQL databases, Lan *et al.* [14] provide a general-purpose anomaly detection approach that relies on features selection to enhance prediction efficiency. Similarly, Guan and Fu [12] perform feature extraction based on PCA to identify the most relevant inputs for anomaly detection. Yet, results of our previous work [21] confirm that a supervised method with all features outperforms a unsupervised one by reducing the number of false positives by 10 %. In this work, we extended our previous supervised learning model to detect multiple classes of anomalies based on SLO metrics.

Guan *et al.* [11] implement a probabilistic prediction model based on a *supervised learning method*. Although their model allows us to compare the dependability of virtualized and non-virtualized cloud systems, it suffers from poor prediction efficiency when it is used to predict cloud performance anomalies. Tan *et al.* [24] propose general-purpose prediction model to prevent performance anomalies. Their *supervised learning*-based model combines 2-dependent Markov chain model with the tree-augmented Bayesian networks. But, the authors did not provide information about the prediction efficiency and the capacity of their approach to generalize. We show with Tejo that the choice of the learning algorithm and features contribute to enhance predictive efficiency of performance anomalies.

6 Conclusion

The emerging stream processing platforms rely on NewSQL databases deployed on the cloud to compute big data with high velocity. However, performance anomalies caused by faults on the cloud infrastructure, that are likely to be common, may undermine the capacity of NewSQL databases to handle fast data processing. To analyse these performance anomalies, we proposed Tejo, a supervised anomaly detection scheme for NewSQL databases. This scheme allows us to evaluate the performance of NewSQL database as faults on network, memory, CPU, and disk occur. Experiments with VoltDB, a prominent NewSQL database, showed that the 99^{th} percentile latency soars two orders of magnitude as memory and network faults happen. We showed that Tejo also provides a learning model to detect these performance anomalies. Our findings suggest that learning algorithms based on boosting methods are better to detect anomalies on a VoltDB cluster, and features from the TCP layer of VMs are the best

predictors. Results also suggest that the contribution of VoltDB-specific features is negligible, therefore our learning model is likely to have similar efficiency with different NewSQL databases.

Acknowledgments. This research is partially funded by the project Secured Virtual Cloud (SVC) of the French program Investissements d'Avenir on Cloud Computing.

References

1. Bonvin, N., Papaioannou, T.G., Aberer, K.: A self-organized, fault-tolerant and scalable replication scheme for cloud storage. In: SoCC (2010)
2. Breiman, L.: Random forests. Mach. Learn. **45**(1), 5–32 (2001)
3. Carbone, M., Rizzo, L.: Dummynet revisited. In: ACM SIGCOMM (2010)
4. Cetintemel, U., Du, J., Kraska, T., Madden, S., Maier, D., Meehan, J., Pavlo, A., Stonebraker, M., Sutherland, E., Tatbul, N., et al.: S-store: a streaming newsql system for big velocity applications. In: VLDB (2014)
5. Chandola, V., Banerjee, A., Kumar, V.: Anomaly detection: a survey. ACM CSUR **41**(3), 15 (2009)
6. Chen, H., Jiang, G., Yoshihira, K.: Failure detection in large-scale internet services by principal subspace mapping. TKDE **19**(10), 1308–1320 (2007)
7. Cortes, C., Vapnik, V.: Support-vector networks. Mach. Learn. **20**(3), 273–297 (1995)
8. Do, T., Hao, M., Leesatapornwongsa, T., Patana-anake, T., Gunawi, H.S.: Limplock: understanding the impact of limpware on scale-out cloud systems. In: SoCC (2013)
9. Eriksson, B., Durairajan, R., Barford, P.: Riskroute: a framework for mitigating network outage threats. In: CoNEXT (2013)
10. Friedman, J.H.: Recent advances in predictive (machine) learning. J. Classif. **23**(2), 175–197 (2006)
11. Guan, Q., Chiu, C.-C., Fu, S.: Cda: a cloud dependability analysis framework for characterizing system dependability in cloud computing infrastructures. In: PRDC (2012)
12. Guan, Q., Fu, S.: Adaptive anomaly identification by exploring metric subspace in cloud computing infrastructures. In: SRDS (2013)
13. Gujrati, P., Li, Y., Lan, Z., Thakur, R., White, J.: A meta-learning failure predictor for blue gene/l systems. In: ICPP (2007)
14. Lan, Z., Zheng, Z., Li, Y.: Toward automated anomaly identification in large-scale systems. In: TPDS (2010)
15. Leskovec, J., Rajaraman, A., Ullman, J.D.: Mining of Massive Datasets. Cambridge University Press, Cambridge (2014)
16. Love, B.C.: Comparing supervised and unsupervised category learning. Psychonomic Bulletin and Review (2002)
17. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: machine learning in python. In: JMLR (2011)
18. Petersen, K., Spreitzer, M., Terry, D., Theimer, M.: Bayou: replicated database services for world-wide applications. In: ACM SIGOPS (1996)

19. Ren, K., Thomson, A., Abadi, D.J.: Lightweight locking for main memory database systems. In: VLDB (2012)
20. Schurman, E., Brutlag, J.: The user and the business impact of server delays, additional bytes, and http chunking in web search (2009)
21. Silvestre, G., Sauvanaud, C., Kaâniche, M., Kanoun, K.: An anomaly detection approach for scale-out storage systems. In: IEEE SBAC-PAD (2014)
22. Stonebraker, M.: Sql databases v. nosql databases. ACM Commun. **53**(4), 10–11 (2010)
23. Stonebraker, M., Weisberg, A.: The voltdb main memory dbms. Data Eng. **36**(2), 21–27 (2013)
24. Tan, Y., Nguyen, H., Shen, Z., Gu, X., Venkatramani, C., Rajan, D.: Prepare: predictive performance anomaly prevention for virtualized cloud systems. In: ICDCS (2012)
25. Wang, A., Venkataraman, S., Alspaugh, S., Katz, R., Stoica, I.: Cake: enabling high-level slos on shared storage systems. In: SoCC (2012)

Resiliency Variance in Workflows with Choice

John C. Mace(✉), Charles Morisset, and Aad van Moorsel

School of Computing Science, Newcastle University,
Newcastle upon Tyne, NE1 7RU, UK
{john.mace,charles.morisset,aad.vanmoorsel}@ncl.ac.uk

Abstract. Computing a user-task assignment for a workflow coming with probabilistic user availability provides a measure of completion rate or *resiliency*. To a workflow designer this indicates a risk of failure, especially useful for workflows which cannot be changed due to rigid security constraints. Furthermore, resiliency can help outline a *mitigation strategy* which states actions that can be performed to avoid workflow failures. A workflow with choice may have many different resiliency values, one for each of its execution paths. This makes understanding failure risk and mitigation requirements much more complex. We introduce *resiliency variance*, a new analysis metric for workflows which indicates volatility from the resiliency average. We suggest this metric can help determine the risk taken on by implementing a given workflow with choice. For instance, high average resiliency and low variance would suggest a low risk of workflow failure.

Keywords: Workflow satisfiability problem · Quantitative analysis · Resiliency metrics

1 Introduction

Many business domains including finance, healthcare and eScience use the concept of workflow to efficiently orchestrate their everyday business processes [5, 14,15]. Although definitions may vary, workflows typically consist of tasks (*the work*) and ordering conditions (*the flow*) [1]. Completing every execution, or instance of a workflow means assigning each task to a user in accordance with required security constraints. Often these are enforced by regulations ensuring only users with correct capabilities are matched with appropriate tasks whilst limiting data access and reducing the threat of collusion and fraud [7,17].

Finding a user assignment for every task such that all security constraints are met is a well studied problem, known as the *workflow satisfiability problem* (WSP) [9,29]. The WSP has been shown to be NP-hard, meaning every combination of users to tasks may have to be tried before finding an assignment that satisfies a workflow. The WSP assumes all users will be available during execution, however periodic user unavailability at runtime means a satisfiable workflow at design time may become unsatisfiable during its operation.

A. Fantechi and P. Patrizio (Eds.): SERENE 2015, LNCS 9274, pp. 128–143, 2015.
DOI: 10.1007/978-3-319-23129-7_10

In cases where no valid user is available for a specific task assignment, the security constraints inadvertently block a workflow from completing. Any available users are either not permitted to perform the task, or are permitted but cannot do so due to constraints with previously executed tasks. Without violating the security policy, the alternative is to terminate early thus causing a workflow *failure*. Forcing early termination of a workflow may bring heavy operational penalties in terms of monetary costs, lost productivity and reduced reputation. In practice, blocked workflows are typically managed by performing mitigating actions which facilitate a completable workflow, often essential in healthcare and other critical domains where failure tolerance is small. For example, it may be that authorising a security override (e.g. break glass [24]) has less long-term impact than allowing the workflow to fail. Elucidating permitted mitigation actions to be taken if a workflow becomes blocked forms a workflow *mitigation strategy*.

When designing workflows it is favourable to predict the risk of workflow failure and understand requirements, in terms of actions, impact and cost of a suitable mitigation strategy. This is especially important for workflows coming with rigid security constraints that cannot be changed at design time. One method is to consider the *workflow resiliency problem*, an extension of the WSP that looks to find an assignment to satisfy a workflow even when some users become unavailable [29]. The quantitative approach to this problem taken in [20] allows a workflow's resiliency to be expressed as a measure of expected completion rate. This value in turn indicates the risk of workflow failure, and therefore the likely need to perform mitigation actions.

In [20], the authors consider analysing the resiliency of workflows with only sequential and parallel control patterns such that each has a single execution path. Computing the resiliency for workflows of this form provides a singular comprehensible indicator of failure risk. Low failure risk (high resiliency) would imply an infrequent need to perform any mitigation actions. This could favour a mitigation strategy consisting of short-term, low cost actions such as a security constraint emergency override. High failure risk (low resiliency) would suggest a broader strategy including more permanent yet costly mitigation actions such as staff training and repealing user unavailability.

This paper considers workflows with gateways, or choice co-ordinators such that multiple execution paths exist that can be taken at runtime to complete a workflow, and where each path may come with a different resiliency value. Understanding risk failure and mitigation strategy requirements of such workflows can be much more complex, especially when a workflow contains hundreds if not thousands of execution paths. Taking the resiliency average, or *expected resiliency* alone may be a misleading indicator of failure risk, especially when a workflow contains paths of both very high and very low resiliency.

We introduce *resiliency variance*, a new metric for workflow failure risk analysis that indicates overall resiliency variability or *volatility* from the resiliency average. In business terms, volatility is typically viewed as a measure of risk; a variance metric helps determine the risk an investor might take on when purchasing a specific asset [11]. Similarly, resiliency variance could provide a workflow

designer with an indicator of failure risk taken on by implementing a given workflow with choice. This could also be useful for predicting a suitable mitigation strategy. For example, a workflow with high expected resiliency and low variance indicates low failure risk and mitigation cost whilst high variance would suggest a much higher failure risk and mitigation cost.

We give an overview of workflow resiliency related work in Sect. 2 whilst Sect. 3 defines a workflow with choice. Section 4 discusses workflow resiliency and its calculation before introducing resiliency variance and show how it is calculated using a real-world university based purchase request workflow. Section 5 provides a discussion on workflow mitigation techniques and how resiliency variance could inform mitigation strategy choice. Concluding remarks are given in Sect. 6.

2 Related Work

A number of previous studies on workflow resiliency and its enhancement appear in the literature. Wang et al. took a first step in [29] to quantify resiliency and declare a workflow as k resilient if it can withstand up to k absent users in all instances. In [20] Mace et al. consider workflows that are not always k resilient and provide a measure of quantitative resiliency indicating how much a workflow is likely to terminate for a given security policy and user unavailability model. This approach illustrates a trade-off exists between aspects such as success rate, expected termination point and computation time.

Basin et al. in [4] overcome scenarios where no valid user-task assignment exists by reallocating roles to users at runtime to satisfy security constraints. A new assignment of users to roles is calculated with the minimum cost to risk, administration and maintenance. This is feasible in certain business domains but may have limited application in workflows where roles are more specialised; for example is adding an untrained user to the role doctor to satisfy a security policy better than overriding it and enabling a constrained but qualified doctor?

Wainer et al. consider in [28] the explicit overriding of security constraints in workflows, by defining a notion of privilege. In [8] Brunel et al. suggest a security policy may still be satisfied even though some security constraints may be violated. This is considered acceptable by defining additional conditions that apply in the case of violation that must be satisfied to comply with the security policy. Bakkali [2] suggests enhancing resiliency through delegation and the placement of criticality values over workflows. Delegates are chosen on their suitability but may lack competence; this is considered the 'price to pay' for resiliency. As delegation takes place at a task level it is not currently clear whether a workflow can still complete while meeting security constraints. In [10] Crampton et al. suggest a mechanism that can automatically respond to the absence of users by delegating a task appropriately when no qualified user is available to perform it.

Current literature does not fully address the issue of workflows that must operate but may not be resilient in every instance. Although many approaches have been suggested in isolation, a range of different remediation options including policy overrides are necessary for a more optimal solution.

3 Workflow

In general, a workflow consists of a set of tasks which can be executed following some constraints: some tasks must be executed before others, some tasks can be executed in parallel, some tasks can be executed instead of others. There exist several definitions for workflows in the literature, for instance as a partial ordering of tasks [9] or as a directed graph [27]. The aim of the work presented here is to study the resiliency of workflows with choice, using the notion of resiliency introduced in [20], where a workflow is defined as a set of users, a partially ordered set of tasks and a security policy.

However, this definition does not allow for *choice*, i.e., for having different paths in a workflow according to the evaluation of some choices. For instance, a workflow managing the purchasing process might have different tasks based on the cost of the purchase. In this section, we first give an inductive definition for a workflow with choice inspired from [16], and we show how it can be reduced into a definition compatible with [20].

3.1 Task Structure with Choice

A *task structure* is built upon two sets: a set T of atomic tasks and a set C of atomic choices. Intuitively, the former set represents each action that can be performed, while the latter represents the different points where the workflow can branch. The set TSC of task structures with choice is then defined inductively:

- Given a single task $t \in T$, t also belongs to TSC;
- Given two task structures $ts_1 \in TSC$ and $ts_2 \in TSC$, $ts_1 \rightarrow ts_2$ also belongs to TSC, and corresponds to the sequential execution of ts_1 followed by ts_2;
- Given two task structures $ts_1 \in TSC$ and $ts_2 \in TSC$, $ts_1 \wedge ts_2$ also belongs to TSC, and corresponds to the parallel ordering ts_1 and ts_2;
- Given a choice $c \in C$ and two task structures $ts_1 \in TSC$ and $ts_2 \in TSC$, $c : ts_1 \,?\, ts_2$ also belongs to TSC, and corresponds to the task structure ts_1 if c evaluates to true, and to ts_2 otherwise.

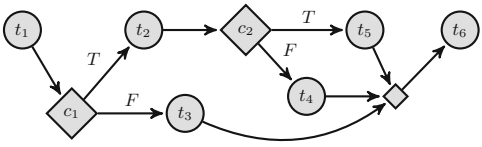


Fig. 1. Running example task structure

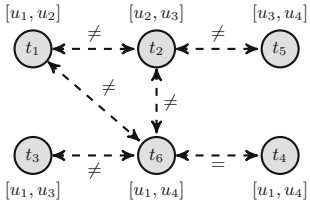


Fig. 2. Running example security policy

Running example. *As a running example to illustrate the different concepts presented here, we define* $T = \{t_1, t_2, t_3, t_4, t_5, t_6\}$, $C = \{c_1, c_2\}$ *and*

$$ts_1 = t_1 \rightarrow [c_1 : [t_2 \rightarrow [c_2 : t_5 \,?\, t_4]] \,?\, t_3] \rightarrow t_6$$

Note that for the sake of simplicity, we do not consider in the running example any parallel composition. We give a graphical representation of ts_1 *in Fig. 1 where tasks are represented as circles and choices as diamonds. In order to represent the end of a choice, we use the empty diamond symbol, and in this particular example, both choices* c_1 *and* c_2 *finish at the same point. The directed arcs represent the ordering of task execution.*

It is worth pointing out that in the graphical notation used in Fig. 1, the choice nodes correspond to or-nodes and the empty diamond to a merge coordinator in [27].

3.2 Task Structure Reduction

At runtime, the choices in a task structure are resolved, and only the corresponding paths are executed. We adopt here an approach where we do not know how each choice is going to be resolved at runtime, and we therefore consider beforehand all possible solutions. Intuitively, we want to *reduce* a task structure with choice to one without choice, for which all tasks should be executed.

Hence, we write TS for the subset of TSC corresponding to task structures without choice, and we model the reduction process through the function $red\colon TSC \times \wp(C) \rightarrow TSC$, such that, given a task structure ts and a set of choices $\gamma \subseteq C$, $red(ts, \gamma)$ corresponds to the reduction of ts where each choice in γ is evaluated as true, and any other choice as false. More formally:

$$\begin{aligned} red(t, \gamma) &= t \\ red(ts_1 \rightarrow ts_2, \gamma) &= red(ts_1, \gamma) \rightarrow red(ts_2, \gamma) \\ red(ts_1 \wedge ts_2, \gamma) &= red(ts_1, \gamma) \wedge red(ts_2, \gamma) \\ red(c : ts_1 \,?\, ts_2, \gamma) &= \begin{cases} red(ts_1, \gamma) & \text{if } c \in \gamma \\ red(ts_2, \gamma) & \text{otherwise} \end{cases} \end{aligned}$$

All possible instances without choice of a task structure with choice can be defined by:

$$ins(ts) = \{ts' \in TS \mid \exists \gamma \subseteq C \; red(ts, \gamma) = ts'\}$$

A task structure without choice can be converted to a set of tasks with a partial ordering, thus allowing us to reuse existing corresponding techniques. Given a task structure ts, we first write $\tau(ts)$ for the set of tasks appearing in ts (which can be straightforwardly defined by induction over ts). We then define the function $ord : TS \rightarrow \wp(T \times T)$, which, given a task structure without choice

ts, returns the ordering relation over the tasks in ts.

$$\begin{aligned} ord(t) &= \emptyset \\ ord(ts_1 \wedge ts_2) &= ord(ts_1) \cup ord(ts_2) \\ ord(ts_1 \rightarrow ts_2) &= \{(t_1, t_2) \mid t_1 \in \tau(ts_1) \wedge t_2 \in \tau(ts_2)\} \\ &\quad \cup ord(ts_1) \cup ord(ts_2) \end{aligned}$$

Running example. *The possible instances of ts_1 are:*

- $t_1 \rightarrow t_2 \rightarrow t_5 \rightarrow t_6$ *(corresponding to $\gamma = \{c_1, c_2\}$);*
- $t_1 \rightarrow t_2 \rightarrow t_4 \rightarrow t_6$ *(corresponding to $\gamma = \{c_1\}$);*
- $t_1 \rightarrow t_3 \rightarrow t_6$ *(corresponding to $\gamma = \{c_2\}$ and $\gamma = \emptyset$).*

Since these instances do not contain any parallel structure, the ordering for each instance is simply the total ordering of the tasks following the sequence.

3.3 Security Policy

Next we define a set of users U that comes with a security policy over the set of tasks T. In general, a security policy is a triple $p = (P, S, B)$ where:

- $P \subseteq U \times T$ are *user-task permissions*, such that $(u, t) \in P$ if, and only if u is allowed to perform t.
- $S \subseteq T \times T$ are *separations of duty*, such that $(t, t') \in S$ if, and only if the users assigned to t and t' are distinct.
- $B \subseteq T \times T$ are *bindings of duty*, such that $(t, t') \in B$ if, and only if the same user is assigned to t and t'.

A workflow therefore consists of a set of tasks, with an ordering relation over the tasks, a set of users, and a security policy.

Definition 1. *A workflow is a tuple $w = (ts, U, p)$, where ts is a task structure, U is a set of users, and p is a security policy.*

Note we assume ts to be equivalent to an inducement of the task manager τ given an initial task $t_0 \in T$ from the definition of workflow given in [20]. Given a workflow $w = (ts, U, p)$ and a set of choices $\gamma \subseteq C$, we abuse the notation and write $red(w, \gamma)$ for the workflow $w' = (red(ts, \gamma), U', p')$, where p' corresponds to p restricted to tasks appearing in $red(ts, \gamma)$ and U' corresponds to U restricted to users appearing in p'. Similarly, we write $ins(w)$ for the set of workflows w' such that there exists $\gamma \subseteq C$ satisfying $w' = red(w, \gamma)$.

Running example. *We now consider a set of users $U_1 = \{u_1, u_2, u_3, u_4\}$ and a security policy $p_1 = (P_1, S_1, B_1)$ that states:*

- $P_1 = \{(u_1, t_1), (u_2, t_1), (u_2, t_2), (u_3, t_2), (u_1, t_3),$
  $(u_3, t_3), (u_1, t_4), (u_4, t_4), (u_3, t_5), (u_4, t_5), (u_1, t_6),$
  $(u_4, t_6)\}$

- $S_1 = \{(t_1, t_2), (t_1, t_6), (t_2, t_5), (t_2, t_6), (t_3, t_6\}$
- $B_1 = \{(t_4, t_6)\}$.

Figure 2 illustrates p_1, where the dotted arrows labelled '$\neq$' and $=$ signify the constraints given in S_1 and B_2 respectively. A label $[u_m, \ldots, u_n]$ states the users that are authorised by P_1 to execute t_i.

4 Workflow Resiliency

Given a workflow $w = (ts, U, p)$, we need to assign tasks in ts to users in U in order to execute them, while respecting the policy p. If ts contains some choice elements, it is not strictly necessary to assign all tasks, only those that will be chosen at runtime. However, as mentioned above, we assume here that we have no control over the choices, and therefore we cannot know beforehand which subset of tasks must be assigned. Hence, we reduce the problem of task assignment for a workflow with choice to considering the task assignment of all possible instances without choice, thanks to the function *red*. In this section, we first describe the *resiliency problem* for workflows without choice, following the existing literature, and we then lift the problem to workflows with choice.

4.1 Resiliency Without Choice

Given a workflow without choice, finding a complete assignment that satisfies all the security constraints is known as the Workflow Satisfiability Problem (WSP), and we refer for instance to [9,29] for further reading on this problem.

Solving the WSP assumes any $u \in U$ will always be available for every instance of a workflow. However in practice, sickness, vacation, heavy workloads, etc., can cause users to periodically be unavailable for task assignments. It is then important to find a valid and complete assignment that maximises the chance of a workflow w to finish: finding an assignment such that w will likely finish 9 out of 10 cases is clearly better than choosing one where w will likely finished only 1 out of 10 cases. This is called the *resiliency problem*, whether a workflow w can be satisfied even when some users become absent.

User unavailability in workflows was introduced by Wang and Li [29], who considered a somewhat *binary* approach where users are either available or not. A workflow is classified as k resilient if the workflow can still be satisfied regardless of which k users become absent. In [20], Mace et al. introduced probabilistic user availability and showed that computing the optimal policy of a Markov Decision Process (MDP [6]) is equivalent to finding an assignment that maximises a value function returning the probability of the workflow w to finish. We refer to [20] for the detail of this approach, and given a workflow without choice w, we write $res(w) \in [0, 1]$ for its resiliency. Our main contribution in this paper consists in adapting this measure to workflows with choice.

It is worth pointing out that understanding when users will and will not be available is an obvious requirement when calculating resiliency, which may

Table 1. Running example probabilistic user availability models

	AM_1				AM_2			
	u_1	u_2	u_3	u_4	u_1	u_2	u_3	u_4
t_1	0.95	0.90	0.96	0.94	0.95	0.90	0.96	0.94
t_2	0.88	1.00	0.90	0.97	0.88	1.00	0.90	0.97
t_3	0.85	0.77	0.99	0.89	0.85	0.77	0.85	0.89
t_4	0.40	0.88	0.89	0.52	0.78	0.88	0.89	0.80
t_5	0.93	0.87	0.96	0.96	0.93	0.87	0.82	0.82
t_6	0.98	0.94	0.98	0.98	0.98	0.94	0.98	0.98

Table 2. Running example resiliency measures

	AM_1	AM_2
res(w_{11})	0.89	0.76
res(w_{22})	0.48	0.74
res(w_{33})	0.92	0.79
$expR(w_1)$	0.76	0.76
$varR(w_1)$	0.0403	0.0004

be deduced from a mixture of operational logs, behavioural analysis and user submissions (known and tentative absences). A key influential aspect is *how* the unavailability of users is modelled in the corresponding MDP [21]. In this paper we consider a dynamic user availability model meaning any user who becomes unavailable for a task may become available again at any step later in the workflow.

A key influential aspect is *how* the unavailability of users is modelled in the corresponding MDP [21]. We assume a dynamic availability model for the rest of this paper.

4.2 Resiliency with Choice

We now consider adapting the resiliency measure for a workflow without choice to a resiliency measure for a workflow with choice using the aid of our running example.

Running example. *We consider a workflow* $w_1 = (ts_1, U_1, p_1)$ *such that* $ins(w_1) = \{w_{11}, w_{22}, w_{33}\}$ *where:*

- $w_{11} = (t_1 \rightarrow t_2 \rightarrow t_5 \rightarrow t_6, U_{11}, p_{11})$
- $w_{22} = (t_1 \rightarrow t_2 \rightarrow t_4 \rightarrow t_6, U_{22}, p_{22})$
- $w_{33} = (t_1 \rightarrow t_3 \rightarrow t_6, U_{33}, p_{33})$.

We consider two different probabilistic user availability models AM_1 *and* AM_2*, given in Table 1 and assume that* AM_2 *is the result of escalation type mitigation actions carried out on* AM_1*, for instance by cancelling user vacations (see Sect. 5.1). An entry* $t_i \times u_i$ *is the probability of user* u_i *being available for the assignment of task* t_i*. The resiliency of each* $w_{ii} \in ins(w_1)$ *is given in Table 2.*

Resiliency Extrema. Finding the minimal resiliency for a workflow with choice w indicates which $w' \in ins(w)$ will give the lowest success rate for w if executed. This can be interpreted as the worst case, or the instance in w with the highest

failure risk. On first glance this indicates which parts of w need the most attention in terms of mitigation. For instance, in our running example, w_1 is most likely to fail when w_{22} is executed which gives the minimal resiliency, 0.48 and 0.74 under AM_1 and AM_2 respectively. Imagine now that under AM_1, w_{22} has a low probability of execution, e.g., 0.01, or 1 execution in 100 cases whereas w_{33} with 0.92 resiliency has a high execution probability, e.g., 0.80, or 80 in 100 cases. In general, the resiliency for w_1 will therefore be much higher meaning a costly mitigating strategy for the infrequent, low resiliency case may not be cost effective.

A bound on the expected success rate can be placed on w by calculating both the maximal and minimal resiliency for w. In our running example under AM_1, w_1 has a large bound with an expected finish rate of between 0.48 (w_{22}) and 0.92 (w_{33}). Under the mitigated AM_2, w_1 has a much smaller bound such that the expected finish rate is between 0.74 (w_{22}) and 0.79 (w_{33}). The resiliency bound can be a useful resiliency measure when all $w' \in ins(w)$ have an equiprobable chance of being executed. If however under AM_1, w_{33} has a low execution probability of 0.01 whilst w_{22} has a high execution probability of 0.8 then in general the resiliency achieved will tend towards the minimal value of 0.48. Placing a bounds on the resiliency in this case becomes a misleading measure of resiliency to the workflow designer.

Resiliency Distribution. Given a workflow with choice w, calculating the resiliency for every possible instance $w' \in ins(w)$ provides the full resiliency distribution for w. This can enable the workflow designer to identify instances of low resiliency, and therefore those needing more extensive mitigation. A tolerance threshold for resiliency may exist for w, deemed acceptable when every instance w' has a resiliency equal to or more than the threshold, in other words the probability that every w' meets the threshold is 1.

In our running example we assume a resiliency threshold of 0.50 and for simplicity, an equiprobable execution model for all $w' \in ins(w_1)$ where the execution probability of w' is 0.33. A more complex probabilistic model could easily be imagined, and we leave such cases for future works. Under AM_1 the probability of w_1 meeting this threshold is therefore 0.66 (*unacceptable*), whilst under the mitigated AM_2 the probability is now 1 (*acceptable*).

Illustrating a comparison of risk failure between a pre and post mitigated workflow to business leaders using resiliency distribution may be complex, especially when they contain hundreds if not thousands of execution paths. It may be more useful for a workflow designer to provide a singular, easy to understand measure of resiliency for a workflow with choice.

Expected Resiliency. We now assume a probability function $prob : W \rightarrow [0, 1]$, which given a workflow without choice $w' \in ins(w)$, returns the probability of w' being executed. The expected resiliency indicates the likely success rate across every instance in a workflow with choice w, calculated as the average resiliency of all $w' \in ins(w)$. We define the function $expR : W \rightarrow [0, 1]$, which

given a workflow with choice w returns the expected resiliency of w.

$$expR(w) = \sum_{w' \in ins(w)} prob(w').res(w')$$

In our running example, assuming $prob(w') = 0.33$ for all $w' \in ins(w_1)$, the expected resiliency is 0.76 for w_1 under both AM_1 and AM_2, shown in Table 2.

This in turn indicates an expected failure rate for w_1 of 0.24. Under AM_1 with an equiprobable execution model means the expected resiliency of 0.76 is not assured with every execution of w_1. Each time the instance w_{22} is executed, the probability of w_1 terminating successfully is only 0.48. This means roughly half of these instance executions will cause w_1 to fail. When executing w_{11} and w_{33} the actual resiliency is much higher than the expected value. Clearly in this case the expected resiliency alone gives a misleading measure of resiliency for a workflow with choice, in other words the expected resiliency cannot actually be *expected* in every case.

Under the mitigated model AM_2, the expected resiliency is now roughly attained whichever $w' \in ins(w_1)$ is executed. In this case the expected resiliency measure alone is arguably enough to indicate the true failure risk of w_1. In other words, a resiliency of ≈ 0.76 can be expected with every execution of w_1. This remains so even when the probabilistic execution model for all $w' \in ins(w_1)$ is not equally weighted. Note that to achieve this the resiliency of w_{11} and w_{33} under AM_1 has been reduced to 0.76 and 0.79 respectively.

Resiliency Variance. The resiliency variance is a measure of how spread out a distribution is, or the variability from the expected resiliency of all instances in a workflow with choice w. A resiliency variance value of zero indicates that the resiliency of all $w' \in ins(w)$ are identical such that the expected resiliency alone will give a true indicator of risk failure. All resiliency variances that are non-zero will be positive. A large variance indicates that instances are far from the mean and each other in terms of resiliency, whilst a small variance indicates the opposite. As discussed in the Introduction, the resiliency variance can give a prediction of volatility or failure risk to a workflow designer taken on when implementing a particular workflow with choice. To quantify the resiliency variance measure we define a function $varW : W \to \mathbb{R}$, which given a workflow with choice w returns the resiliency variance of w.

$$varR(w) = \sum_{w' \in ins(w)} prob(w').(res(w') - expR(w))^2$$

The resiliency variance for our running example w_1, calculated under availability models AM_1 and AM_2 is given in Table 2.

An equiprobable execution model is again used for simplicity. Under AM_1 a resiliency variance of 0.0403 is calculated, equivalent to a large standard deviation of 0.20 ($\sqrt{varR(w_1)}$). Under the mitigated AM_2 the resiliency variance has been reduced to 0.0004, equivalent to a much smaller standard deviation of 0.02

Table 3. Resiliency measures for purchase request workflow

	AM_3	AM_4
minR(w_2)	0.48	0.55
maxR(w_2)	0.96	0.83
expR(w_2)	0.67	0.67
varR(w_2)	0.0183	0.0052

from the expected resiliency. Here we have a decrease by a factor of 10. Clearly this indicates in this case that all instances of w_1 under AM_2 have a probability of terminating successfully close to the expected resiliency of 0.76.

The former case (AM_1) indicates that instances in w_1 can have a large spread in terms of resiliency despite having the same expected resiliency as the latter case (AM_2) coming with a small spread, or variance. Under AM_1, the results show that instances exist in w_1 with much lower and higher probabilities of terminating successfully than the expected resiliency for w_1. The workflow w_1 can be considered *volatile* or *high risk* as it has a high risk of failing if one such instance with low resiliency is executed. Coupled with expected resiliency, resiliency variance can provide an easy to understand measure of workflow risk failure and allow workflow designers to quickly compare similar complex workflows (e.g., pre and post mitigation) to help them predict a suitable mitigation strategy.

4.3 Purchase Request Workflow

In this section we calculate resiliency measures including resiliency variance for a purchase request workflow w_2 that forms part of a real-life procurement procedure used by a large Australian-based university[1]. The workflow consists of 18 atomic tasks and 5 atomic choices, 4 users, and a security policy with 9 separation of duty constraints. The workflow task structure consists of 18 instances, i.e., 18 possible execution paths.

To calculate the resiliency measures we encode w_2 within the probabilistic model checking tool PRISM, which enables the specification, construction and analysis of probabilistic models such as MDPs [19]. PRISM is an intuitive choice as it can model both probabilistic and non-deterministic choice, and gives an efficient way to solve an MDP. For a systematic encoding of a workflow in PRISM and the mechanisms to compute the resiliency measure we refer the reader to the following technical report [22].

Resiliency measures for w_2 are given in Table 3 under two probabilistic availability models AM_3 and AM_4. We assume AM_4 results from mitigation carried out on AM_3. Measures calculated are minimal and maximal resiliency represented as $minR(w_2)$ and $maxR(w_2)$ respectively, expected resiliency and

[1] http://www.fin.unsw.edu.au/files/PP/Purchase_Order_Procedure.pdf.

resiliency variance. The expected resiliency is the same for w_2 under both AM_3 and AM_4 yet the resiliency variance is reduced by a factor of 3.5 under the latter, indicating w_2 now has a lower risk of failure.

5 Mitigation Strategy

In this section we give an overview of the main techniques discussed in the literature that could be implemented within a workflow mitigation strategy to overcome situations when no valid user-task assignment exists. These mitigation actions are categorised into two classes, long-term actions and emergency actions.

5.1 Long-Term Actions

Long-term actions can help raise the resiliency of a workflow by providing a secure solution that does not involve having to violate the security policy or change the task structure [25,26]. Long-term actions can also often provide a more permanent solution to parts of a workflow that commonly becomes blocked. Long-term actions arguably take time and can be expensive in monetary terms to complete, yet the long-term benefits can be high. Those actions of interest include:

- *suspension:* a workflow is suspended until a user becomes available. This can be appropriate if deadlines are not important and/or there is some assurance of future availability. Essentially a task is assigned to a user and executed when the user becomes available.
- *escalation:* the probability of a valid user being available for a task is increased. A user may be asked to return from vacation or come in on their day off, or they may need to suspend another task they are currently executing. We assume the use of this action in our running example to mitigate user availability model AM_1, thereby creating AM_2.
- *training:* a user's capabilities are raised to an acceptable level before granting permission to perform a task.
- *change policy* [3,4] - security constraints are removed or changed (e.g., reallocating roles) which can take time and may need to be done multiple times if a workflow is to complete. Changes may not be possible due to legal requirements or impractical if users do not have the correct skills.

5.2 Emergency Actions

Emergency actions can help raise the resiliency of a workflow by overriding the security policy or changing the task structure. Such actions provide a *quick-fix* to a workflow that becomes blocked but do not offer any permanent solution to parts of a workflow that commonly becomes blocked. A less secure solution is provided than long-term actions that may also impact the output quality of the workflow if the task structure is indeed changed. Emergency actions are arguably

quick and cheap in monetary terms to complete, yet the long-term benefits can be low. A distinction is made between overriding which implies some control is in place over who and how policies can be broken while violation is unsolicited. Those actions of interest include:

- *delegation* [2,13,18]*:* if user is unavailable they may delegate a task assignment to a peer or subordinate who would not normally be authorised to perform the task. This overrides the user-task permissions but can result in lower standards and higher risk.
- *break glass* [23]*:* certain users are given the right to override a security constraint to gain privileges when the assigned user is unavailable, set up with special accounts. Justification is typically sought after access is granted.
- *skipping:* a task is bypassed and executed at a later time, although out of sequence. This is similar to suspension although other tasks are executed while waiting for a user to become available.
- *forward execution:* the workflow instance is rolled back [12] until another assignment path can be taken which bypasses the invalid user-task assignment.

5.3 Strategy Selection

Implementing a suitable mitigation strategy is important to reduce a workflow's chance of failure, especially one with both a high expected success rate and rigid security constraints. Ultimately a favourable mitigation strategy will give a high expected resiliency and a low resiliency variance. Clearly we are not in a position to state which and when particular mitigation actions should be implemented as part of a mitigation strategy as this is highly context dependent. We do however offer some discussion on this matter and show how the resiliency measures for a workflow with choice discussed in Sect. 4.2 could be useful in this regard.

It may be the case that a mitigation strategy can consist of only long-term actions, especially where security is paramount and no emergency actions are permitted. Alternatively, finishing a workflow in a timely manner may be the priority meaning a mitigation strategy consists of only emergency actions. A third option is a mitigation strategy consisting of both long-term and emergency actions that is fully comprehensive and means the most appropriate option is always available.

Although long-term mitigation actions can be costly in both time and monetary terms, it may be the case that such actions need only be performed once. For instance, training a staff member once for a particular task means they can perform the task in all future executions when necessary. Implementing long-term mitigation actions for all instances of low resiliency would seem a sensible option however if some or all low resiliency instances have a very low probability of execution, this approach may not be cost effective. Emergency actions alone may be acceptable. If on the other hand emergency actions are implemented for an instance with a high probability of execution yet low resiliency it is likely

that these often less secure actions will need to be performed multiple times. Long-term actions may be more appropriate here.

Using the minimum resiliency of a workflow with choice may lead to *over* mitigation, especially if the lowest resiliency instances are infrequently executed. Using the maximum resiliency may produce the opposite effect such that a workflow is *under* mitigated. Workflows with high resiliency variance and low resiliency variance can have the same measure of expected resiliency meaning this measure alone may be misleading. The expected resiliency and resiliency variance together can inform mitigation strategy choice as follows:

- *high resiliency and high variance:* a combination of both action types with a higher proportion of emergency actions
- *low resiliency and high variance:* a combination of both action types with a higher proportion of long-term actions
- *high resiliency and low variance:* emergency actions
- *low resiliency and low variance:* long-term actions.

6 Conclusion

It is important that a workflow designer can predict the risk of failure before implementing a workflow, especially if its design must include rigid security constraints. In [20] the probability of finding a user assignment for all tasks in a workflow without choice provides a measure of completion rate or *resiliency*. We extend this approach by considering workflows with choice which may come with multiple resiliency values, one for each execution path. We consider computing different resiliency measures including *resiliency variance* which indicates volatility from the resiliency average. We suggest this metric can help predict the risk taken on when implementing a given workflow and help determine suitable mitigation actions which should be executed when no valid user assignment exists for a workflow task.

References

1. Workflow Management Coalition: The workflow reference model. In: Lawrence, P. (ed.) Workflow Handbook 1997, pp. 243–293. Wiley, New York (1997)
2. Bakkali, H.E.: Enhancing workflow systems resiliency by using delegation and priority concepts. J. Digit. Inf. Manag. **11**(4), 267–276 (2013)
3. Basin, D., Burri, S.J., Karjoth, G.: Obstruction-free authorization enforcement: aligning security with business objectives. In: Proceedings of the 2011 IEEE 24th Computer Security Foundations Symposium, CSF 2011, pp. 99–113. IEEE Computer Society, Washington, DC (2011)
4. Basin, D., Burri, S.J., Karjoth, G.: Optimal workflow-aware authorizations. In: Proceedings of the 17th ACM Symposium on Access Control Models and Technologies, SACMAT 2012, pp. 93–102. ACM, New York (2012)
5. Basu, A., Kumar, A.: Research commentary: workflow management issues in e-business. Inf. Syst. Res. **13**(1), 1–14 (2002)

6. Bellman, R.: A Markovian decision process. Indiana Univ. Math. J. **6**, 679–684 (1957)
7. Botha, R., Eloff, J.H.P.: Separation of duties for access control enforcement in workflow environments. IBM Syst. J. **40**(3), 666–682 (2001)
8. Brunel, J., Cuppens, F., Cuppens, N., Sans, T., Bodeveix, J.-P.: Security policy compliance with violation management. In: Proceedings of the 2007 ACM Workshop on Formal Methods in Security Engineering, FMSE 2007, pp. 31–40. ACM, New York (2007)
9. Crampton, J., Gutin, G., Yeo, A.: On the parameterized complexity and kernelization of the workflow satisfiability problem. ACM Trans. Inf. Syst. Secur. **16**(1), 4 (2013)
10. Crampton, J., Morisset, C.: An auto-delegation mechanism for access control systems. In: Cuellar, J., Lopez, J., Barthe, G., Pretschner, A. (eds.) STM 2010. LNCS, vol. 6710, pp. 1–16. Springer, Heidelberg (2011)
11. Damodaran, A.: Strategic Risk Taking: A Framework for Risk Management, 1st edn. Wharton School Publishing, Upper Saddle River (2007)
12. Eder, J., Liebhart, W.: Workflow recovery. In: Proceedings of the First IFCIS International Conference on Cooperative Information Systems, 1996, pp. 124–134, June 1996
13. Gaaloul, K., Schaad, A., Flegel, U., Charoy, F.: A secure task delegation model for workflows. In: Second International Conference on Emerging Security Information, Systems and Technologies, 2008, SECURWARE 2008, pp. 10–15, August 2008
14. Georgakopoulos, D., Hornick, M., Sheth, A.: An overview of workflow management: from process modeling to workflow automation infrastructure. Distrib. Parallel Databases **3**(2), 119–153 (1995)
15. Hiden, H., Woodman, S., Watson, P., Cala, J.: Developing cloud applications using the e-science central platform. Philos. Trans. R. Soc. A Math. Phys. Eng. Sci. **371**(1983), 20120085 (2013)
16. Kiepuszewski, B., ter Hofstede, A.H.M., Bussler, C.J.: On structured workflow modelling. In: Wangler, B., Bergman, L.D. (eds.) CAiSE 2000. LNCS, vol. 1789, pp. 431–445. Springer, Heidelberg (2000)
17. Kohler, M., Liesegang, C., Schaad, A.: Classification model for access control constraints. In: IEEE International Performance, Computing, and Communications Conference, 2007, IPCCC 2007, pp. 410–417, April 2007
18. Kumar, A., van der Aalst, W.M.P., Verbeek, E.M.W.: Dynamic work distribution in workflow management systems: how to balance quality and performance. J. Manage. Inf. Syst. **18**(3), 157–193 (2002)
19. 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)
20. Mace, J.C., Morisset, C., van Moorsel, A.: Quantitative workflow resiliency. In: Kutyłowski, M., Vaidya, J. (eds.) ICAIS 2014, Part I. LNCS, vol. 8712, pp. 344–361. Springer, Heidelberg (2014)
21. Mace, J., Morisset, C., van Moorsel, A.: Modelling user availability in workflow resiliency analysis. In: Proceedings of the Symposium and Bootcamp on the Science of Security, HotSoS. ACM (2015)
22. Mace, J.C., Morisset, C., van Moorsel, A.: Impact of policy design on workflow resiliency computation time. Technical Report CS-TR-1469, School of Computing Science, Newcastle University, UK, May 2015

23. Marinovic, S., Craven, R., Ma, J., Dulay, N.: Rumpole: a flexible break-glass access control model. In: Proceedings of the 16th ACM Symposium on Access Control Models and Technologies, SACMAT 2011, pp. 73–82. ACM, New York (2011)
24. Povey, D.: Optimistic security: a new access control paradigm. In: Proceedings of the 1999 Workshop on New Security Paradigms, NSPW 1999, pp. 40–45. ACM, New York (2000)
25. Reichert, M., Weber, B.: Enabling Flexibility in Process-aware Information Systems: Challenges, Methods, Technologies. Springer Science & Business Media, Heidelberg (2012)
26. Russell, N., van der Aalst, W.M.P., ter Hofstede, A.H.M.: Workflow exception patterns. In: Martinez, F.H., Pohl, K. (eds.) CAiSE 2006. LNCS, vol. 4001, pp. 288–302. Springer, Heidelberg (2006)
27. van der Aalst, W.M.P., Hirnschall, A., Verbeek, H.M.W.E.: An alternative way to analyze workflow graphs. In: Pidduck, A.B., Mylopoulos, J., Woo, C.C., Ozsu, M.T. (eds.) CAiSE 2002. LNCS, vol. 2348, pp. 535–552. Springer, Heidelberg (2002)
28. Wainer, J., Barthelmess, P., Kumar, A.: W-RBAC - a workflow security model incorporating controlled overriding of constraints. Int. J. Coop. Inf. Syst. **12**, 2003 (2003)
29. Wang, Q., Li, N.: Satisfiability and resiliency in workflow authorization systems. ACM Trans. Inf. Syst. Secur. **13**(4), 40:1–40:35 (2010)

Author Index