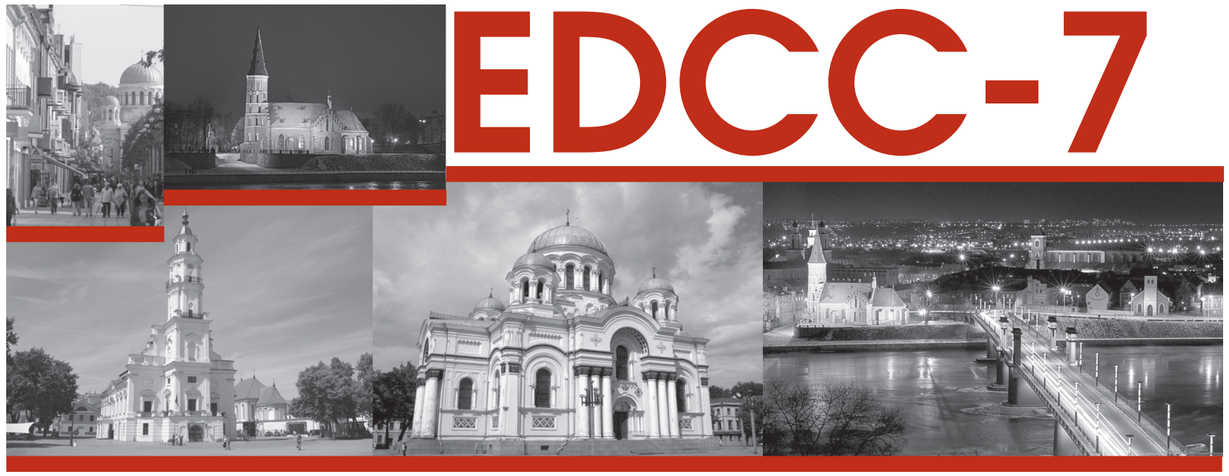
Seventh European Dependable Computing Conference

Seventh European Dependable Computing Conference



EDCC-7

Kaunas, Lithuania 7-9 May 2008

Proceedings Supplemental Volume

Vytautas Magnus University

Proceedings Companion

Seventh European Dependable Computing Conference

EDCC-7

Proceedings Companion

Seventh European Dependable Computing Conference

EDCC-7

7-9 May 2008
Kaunas, Lithuania

Sponsored by:

SEE Working Group "Dependable Computing", France
GI/ITG/GMA TC on Dependability and Fault Tolerance, Germany
AICA Working Group "Dependability of Computer Systems", Italy

In cooperation with:

IFIP Working Group 10.4 "Dependable Computing and Fault Tolerance"
EWICS TC7 on Safety, Reliability and Security
IEEE Computer Society TC on Dependable Computing and Fault Tolerance
Universidad Politecnica de Madrid, Spain
ReSIST, a European Network of Excellence

Hosted by:

Vytautas Magnus University, Kaunas, Lithuania

Vytautas Magnus University

Table of Contents

Proceedings Companion

Seventh European Dependable Computing Conference

EDCC-7 2008

Session 3B: Fast Abstracts III

Dependability Assessment and Analysis

Session 3B: Student Forum

Keynote Speaker

Angelo Marino, Research Programme Officer, European Commission-Information Society and Media DG,Brussels, Belgium

Session 4: Security Issues

Message from the General Chair

It is a pleasure to welcome the participants of EDCC-7 in Kaunas, the second city of the Republic of Lithuania. Kaunas is a university town – we have 12 diverse institutions of higher education with close to 50000 students in the city and its suburbs. The conference is hosted by Vytautas Magnus University. VMU is a general scope university with emphasis on the humanities and social sciences, and with a strong Faculty of Informatics. The University was founded in 1922 as the first university of the Republic of Lithuania that had regained its independence in 1918 after 122 years of captivity within the Russian Empire. After the occupation of Lithuania by the Soviet Union in 1944, the university was closed in 1950. Only the faculties of engineering and medicine were kept functioning as separate Institutes. The reopening of VMU as an autonomous, North American style university took place in 1989, as the Soviet Union was beginning to fall apart. The parliament of Lithuania declared the reestablishment of independence on 11 March 1990, and VMU has grown from 180 first-year students in 1989 to over 8000 students today.

EDCC-7, the Seventh European Dependable Computing Conference is preceded by EDCC conferences held in Berlin (1994), Taormina, Italy (1996), Prague (1999), Toulouse (2002), Budapest (2005), and Coimbra, Portugal (2006). We look forward to continuing the tradition of lively and informative meetings in Kaunas in May of 2008. We hope that the participants will take some time to visit Kaunas and the capital city of Vilnius after the conference. A travel agency will be available to arrange such tours – please stay for the weekend and enjoy Lithuanian hospitality a little longer.

The program of EDCC-7 consists of the 17 full-length papers contained in this volume, the panel discussion “How Hard Is Assessing and Measuring Resilience?”, the Keynote panel “Future Challenges to Dependable and Secure Computing”, the Keynote address by Mr. Angelo Marino , Research Programme Officer of the European Commission, and over 20 short papers called “Fast Abstracts” that describe work in progress or state positions on new or controversial issues. Texts of all presentations that are not contained in this volume will be published in the Supplemental Volume of EDCC-7 Proceedings.

The organization of EDCC-7 has required the dedicated volunteer efforts of many colleagues in the fields of dependable and secure computing. My most sincere thanks go to Ricardo Jiménez-Peris, the Chair of the Program Committee and all its members, and to Ernesto Jiménez-Merino, who edited this volume as well as the Supplemental volume of the Proceedings. Thanks to all members of the Organizing Committee whose names appear in the following Program Chair’s message, and to my VMU colleagues Juozas Augutis and Kęstutis Šidlauskas who took on the local arrangements task. Mr. Valdas Kubilius, the Director of the World Lithuanians’ Center and his staff are managing the registration and social events. Thanks to them for taking on the task on very short notice. The Chair of the Steering Committee, my dear colleague Luca Simoncini, was most helpful in many aspects of organizing EDCC-7, including the job of convincing me to take on the General Chair’s position. Should I thank him for that? That remains to be seen on May 10.

Finally, EDCC-7 has a new and very informative add-on feature. In the morning of May 9 there will be presented a comprehensive overview of the results thus far obtained by the three-year (2006-2008) EC 6th Framework project *ReSIST – Resilience for Survivability in IST* . ReSIST is a Network of Excellence with 18 academic and industrial partners from 8 countries in Europe. In addition to this overview session, members of the ReSIST project are also presenting regular and short papers and taking part in both panels. Please stay for the free ReSIST session and enjoy an advanced look at their results. Many thanks to ReSIST Coordinator Jean-Claude Laprie and to all his project colleagues for their most valuable contribution.

Algirdas Avižienis
General Chair of EDCC-7

Message from the Program Committee Chair

Welcome to the Seventh European Dependable Computing Conference (EDCC-07) hosted at Kaunas, Lithuania, during May 7-9, 2008. The conference program covers a broad range of topics in the different areas of dependability showing the depth and breadth of our community.

The conference attracted 60 submissions from Europe, America and Asia. The PC meeting was held in Madrid in December 2007. It had a significant attendance with over 3/4 of the PC members present at the meeting. The PC members had a very active and constructive participation in the discussion of the papers. This year the selection process was very competitive resulting in selecting 13 regular papers out of 50 (26%). Each paper was assigned to 4 PC members and 97% of the papers got 4 reviews and the rest 3 reviews, yielding an average of almost 4 reviews per paper. The PC also selected 4 submissions in the category of practice experience reports and tool/prototype descriptions and one panel. The program is complemented with two keynote speeches.

Once again I would like to express my most sincere gratitude to the members of the Program Committee who did an excellent work with their thorough reviews and lively discussing the papers during the PC meeting. I appreciate very much that they accepted my invitation to serve in the Program Committee despite their busy schedules.

Many individuals helped me in many different ways during the long process that produced this program. I would like to especially thank Luca Simoncini in his role as Steering Committee Chair and Algirdas Aviezienis in the role as General Chair for his wise advice, support, and mentoring, Johan Karlson PC chair of EDCC-06 for sharing with me his experience in the former PC. My gratitude also goes to Ernesto Jimenez for his help during the PC meeting and his work as publication chair. I would also like to thank Leonardo Querzoni and Jose Orlando Pereira for their work in chairing the Student Forum and Fast abstract tracks. Also my thanks to Sasha Romanovsky for his great job in publicizing the call for papers and call for participation of the different conference tracks. My thanks also to Lorenzo Alvisi, Francisco Brasileiro, and Xavier Defago for disseminating the call for papers in North America, South America, and Asia. Finally, my thanks to Andrea Bondavalli and Henrique Madeira for presenting and organizing the panel.

Ricardo Jiménez-Peris
Program Chair EDCC-07

Message from the Fast Abstracts Chair

Fast Abstracts are short presentations aimed at reporting on current work that may or may not be complete, introducing new ideas to the community, or stating positions on controversial issues or open problems. They are not subject to a formal refereeing process. Instead, they are lightly screened and selected mainly on their relevance and interest to the community.

This year's program features 21 papers on three sessions covering a number of topics of interest to EDCC and the dependability community, namely, on dependability in distributed systems, security and dependability ontologies, and dependability assessment and analysis.

We hope that presentations provide valuable feedback to the authors, provide new insights to the audience, and foster new ideas and collaboration opportunities.

José Orlando Pereira
Fast Abstracts Chair of EDCC-7

Message from the Student Forum Chair

It's my pleasure to welcome you to the Student Forum track of the Seventh European Dependable Computing Conference (EDCC-07) hosted at Kaunas, Lithuania.

The Student Forum is a track were students at an early stage of their research activity can present futuristic, sometimes "off the wall" ideas in the area of Dependable Computing. Aim of this Forum is to provide these students with useful feedback on their work that can help them in further improve early-stage results and collect experience from the comments of more experienced and senior researchers attending their presentations.

Student Forum papers have been selected through a light reviewing process basing on their relevance to the topics of the main conference, and on the interest of the proposed problems. I would like to thank the members of the Student Forum Program Committee for providing their help in this reviewing process.

I hope that this Student Forum will provide valuable feedback to the authors, and foster constructive discussions and new ideas among the attendees.

Leonardo Querzoni
Student Forum Chair of EDCC-7

Organizing Committee

General Chair
Algirdas Avizienis, *Vytautas Magnus University in Kaunas, Lithuania*

Program Chair
Ricardo Jiménez-Peris, *Technical University of Madrid, Spain*

Student Forum Chair
Leonardo Querzoni, *Sapienza University of Rome, Italy*

Fast Abstracts Chair
José Orlando Pereira, *Minho University, Portugal*

Local Arrangements Chair
Juozas Augutis, *Vytautas Magnus University in Kaunas, Lithuania*

Publication Chair
Ernesto Jiménez-Merino, *Technical University of Madrid, Spain*

Finance Chair
Kestutis Sidlauskas, *Vytautas Magnus University in Kaunas, Lithuania*

Publicity Chair
Sasha Romanovsky, *University of Newcastle upon Tyne, United Kingdom*

Steering Committee Chair
Luca Simoncini, *University of Pisa, Italy*

International Liaison Chairs
North America: Lorenzo Alvisi, *University of Texas at Austin, USA*
Latin America: Francisco Brasileiro, *Federal University of Campina Grande, Brazil*
Asia: Xavier Defago, *JAIST, Japan*

Program Committee

Roberto Baldoni, Italy
Angelos Bilas, Greece
Andrea Bondavalli, Italy
Cristian Cachin, Switzerland
Domenico Cotroneo, Italy
Jean-Charles, France
Antonio Fernandez-Anta, Spain
Christof Fetzer, Germany
Roy Friedman, Israel
Felicita di Giandomenico, Italy
Pedro Gil, Spain
Janusz Gorski, Poland
Elena Gramatova, Slovakia
Flavio Junqueira, Spain
Mohamed Kaaniche, France
Johan Karlsson, Sweden
Rogerio de Lemos, UK
Henrique Madeira, Portugal
Erik Mahle, Germany
István Majzik, Hungary
Miroslaw Malek, Germany
Gilles Muller, France
Edgar Nett, Germany
Rui Oliveira, Portugal
Esther Pacitti, France
Marta Patiño, Spain
Fernando Pedone, Switzerland
Peter Popov, UK
Jaan Raik, Estonia
Michel Raynal, France
Luis Rodrigues, Portugal
Luigi Romano, Italy
Juan Carlos Ruiz, Spain
Andre Schiper, Switzerland
Santosh Shrivastava, UK
Matteo Sonza Reorda, Italy
Neeraj Suri, Germany
François Taiani, UK
Elena Troubitsyna, Finland
Paulo Veríssimo, Portugal
Roman Vitenberg, Norway

Student Forum Program Committee

Sergio Mena, University of York, United Kingdom
Corentin Travers, INRIA, France
Paulo Sousa, University of Lisbon, Portugal

EDCC Steering Committee

Luca Simoncini, Italy, Chair
Algirdas Avižienis, Lithuania
Mario Dal Cin, Germany
Karl Grosspietsch, Germany
Karama Kanoun, France
Johan Karlsson, Sweden
Jean-Claude Laprie, France
András Pataricza, Hungary
Brian Randell, UK
João Gabriel Silva, Portugal
Janusz Sosnowski, Poland
Raimund Ubar, Estonia

Session 1B: Fast Abstracts I

Distributed Systems

EDCC-7

Towards Rigorous Engineering of Resilient Pervasive Systems

Alexei Iliasov[1], Linas Laibinis[2], Alexander Romanovsky[1], Kaisa Sere[2], Elena Troubitsyna[2]
[1]Newcastle University, UK, [2]Åbo Akademi University, Finland
{Alexei.Iliasov,Alexander Romanovsky}@ncl.ac.uk,
{Linas.Laibinis,Kaisa.Sere,Elena.Troubitsyna}@abo.fi

Abstract

While pervasive systems offer versatile computing environment, their complexity poses a significant challenge to their developers. Hence ensuring resilience of pervasive systems is an important issue, which should be tackled by adopting rigorous design methods and systems approach. In this short paper we identify the key research directions in engineering pervasive resilient systems and discuss our experience in rigorous development of a multi-agent application called Ambient Campus.

1. Introduction

Pervasive systems should smoothly weave themselves into our everyday life. In general, pervasive systems are complex computing environments integrating a large number of heterogeneous devices. They can be considered as complex decentralized distributed systems composed of loosely-coupled components, which communicate asynchronously while providing system services. Since the system components are typically autonomous and mobile, such systems have to be open and dynamically reconfigurable. Ensuring reliable provision of system services is a challenging task due to component mobility, dynamically changing configuration and inherent unreliability of communicating channels. Hence we should aim at designing resilient systems, i.e., the systems that are able to withstand unpredictable changes and faults in their computing environments.

2. Resilient Pervasive Systems

Developing resilient pervasive systems is a demanding task because the developers of these systems are facing overwhelming complexity coming from various sources. Generally speaking, this complexity is caused by a large size, openness and distributed nature of such systems. Let us observe that even though complexity is perceived as a major threat to *dependability* of computing systems, our society nevertheless puts a high level of *reliance* on pervasive systems, trusting them to perform a wide range of everyday critical functions. These systems are rapidly becoming business- and safety-critical.

Traditionally system complexity is managed via abstract modelling, decomposition and iterative development. In our work we focus on creating models and development techniques for designing resilient pervasive systems. For instance, we specifically focus on explicit modelling of mechanisms for coping with system impairments (faults, errors and failures), since a resilient system needs to be tolerant to the harmful faults, which are most likely to affect the overall system service. Moreover, system fault tolerance should be convincingly demonstrated.

Fault tolerance is an acute issue in designing resilient pervasive systems, since complexity and openness of these systems make it *impossible* to avoid or remove faults altogether. Moreover, fault tolerance is needed to deal with mismatches caused by heterogeneity of components, modes of operation and requirements. Pervasive systems are susceptible to tangled erroneous conditions caused by combinations of errors, their complex interference, as well as errors caused by malfunctioning infrastructures and misbehaving hosts and agents. Furthermore, the growing involvement of the *non-professional* users requires even higher degree of fault tolerance.

Intricacy of pervasive systems makes it extremely difficult for the developers to design appropriate and effective fault tolerance measures. Moreover, the lack of systems view while implementing the fault tolerance mechanisms results in further complication of system design and, as a consequence, makes it even more error-prone.

3. Challenges in Developing Resilient Pervasive Systems

Currently designers often use ad-hoc techniques for engineering pervasive systems that provide quick and

cheap solutions for constructing various "lightweight" applications. However, such techniques cannot ensure the appropriate level of system resilience, especially when used in designing complex business- or safety-critical applications. To guarantee a high degree of resilience of such applications, we need to employ *rigorous methods for system engineering* – the methods supporting a disciplined development process based on formal modelling and reasoning about system correctness. This is why one of the main challenges in engineering complex pervasive systems is the creation of advanced methods and tools for *resilience-explicit* [5] rigorous design. This will include development of the generic design and modelling patterns and tools supporting the rigorous methods.

It is currently widely recognized that only by adopting a *systems approach* [4,10] a high degree of systems resilience can be achieved. Hence, the methods and tools to be defined will need to support a systems approach to development. The essence of a systems approach is in capturing not only the system behaviour but also a relevant part of its environment in the system model. Via a chain of model transformations we arrive at a detailed system model later in the development process. Then we decompose it into the desired components and make the decisions about partitioning into software- and hardware-implementable parts. This approach should be especially suitable for resilience-explicit development of pervasive systems since it allows developers to clearly express the essential properties of the system without imposing constraining design decisions early in the design process.

In particular, we have identified a number of specific issues which should be addressed to create a viable methodology for rigorous engineering of resilient pervasive systems:

- ensuring interoperability of independently developed components during formal construction
- design and formal modelling of advanced mechanisms for tolerating the faults typical for pervasive systems
- development of the advanced application-level fault tolerance techniques, such as flexible exception handling mechanisms suitable for open asynchronous and anonymous systems
- formal modelling and development of resilient context-aware pervasive systems
- rigorous methods for stepwise development of resilient pervasive systems correct-by-construction
- formal modelling of context-awareness
- the rigorous approaches to ensure correctness in the presence of code mobility
- formal modelling of unreliable communication channels and advanced mechanisms for tolerating faults occurring in volatile communication environment
- ensuring correctness in the presence of openness and hostile behaviour
- advanced modelling of self-adaptation and self-protection
- reusable patterns for modelling and development of fault tolerant pervasive systems.

4. Research on Pervasive Systems in RODIN

In our recent work conducted within a FP6 STREP-project on Rigorous Open Development Environment for Complex Systems - RODIN [12], we have addressed the issues identified above. In particular, we have actively worked on creating a methodology for rigorous development of fault tolerant agent systems.

To enable rigorous development of resilient pervasive applications, we created *Context-Aware Mobile Agents* (CAMA) framework [6,7,11] supporting a set of abstractions ensuring system structuring, exception handling and openness. A CAMA system consists of a set of *locations*. Active entities of the system are mobile agents. A CAMA *agent* is a piece of software that conforms to its formal specification. Each agent is executed on its own platform providing an execution environment and interface to the location middleware. A CAMA location is also a container for *scopes*. A scope structures the activity of agents in a specific location and provides an isolation of several communicating agents, thus structuring the communication space. As a result, an agent can cooperate only with the agents participating in the same set of scopes. Moreover, mobile agents can logically and physically migrate from a location to a location.

An agent is built from one or more *roles*. A CAMA *role* is a specification of particular agent functionality, so that composition of specifications of all agent roles forms its specification. In other words, a role is a structuring unit of an agent. An agent participates in a scope by assuming one of the roles available for that scope, which makes a role also an important part of the scoping mechanism. All together, it allows a dynamic composition of multi-agent applications and ensures agent interoperability by enforcing the developers of roles and agents to satisfy the given role specifications.

In RODIN, we have also created a number of abstract reusable patterns for specifying middleware for mobile location-based agent systems. Our patterns are formalized in Event B [2,3,13] – a framework extending the B Method [1] to model complex distributed systems. Event B supports top-down development of systems correct by construction. We start from an abstract specification of the overall agent system, i.e., we abstractly model mobile agents together with a location supporting inter-agent communication. In a number of correctness preserving steps we incorporate various system properties, including fault tolerance, into our formal models. As a result, we arrive at the specification of the entire middleware, which can be then decomposed into separate parts to be implemented by the location and by each individual agent.

In independent development of individual agents, the programmers merely need to augment this abstract part with an implementation of the desired agent functionality. Such an approach allows us to ensure inter-operability of individually developed agents and the correctness of the overall system. Moreover, since the proposed patterns contain abstract specifications of the means for detecting errors as well as the corresponding error recovery procedures, we can guarantee fault tolerance of an agent system developed according to the proposed approach.

Agent systems should operate in volatile error-prone environment. Indeed, mobile agents have to cope with various kinds of communication failures, software failures of other collaborating agents and, of course, their own internal failures. System openness brings even more concerns, such as interoperability, security and trustworthiness. The types of mobile agent system failures can be roughly grouped into the following categories:

1. failures to deliver service by the hosting environment
2. failures in one of the collaborating agents
3. internal agent failures
4. environment failures.

Failures of the first category include all types of transient failures, such as disconnection, migration, spawning, inability to deliver messages, etc. Such failures may be caused by changes in the environment, for example, agent migration, and are better handled by application logic.

Handling the failures from the second category is challenging due to dynamic composition typical for mobile agents. In open systems, each agent has its own interest in cooperation with other agents. Even though this cooperation is established dynamically, there must be some mechanism to encourage communication among matching agents and prevent it among incompatible ones. We also need to create the mechanisms for inter-agent exception propagation and cooperative exception handling. We believe that this is essential for a disciplined and fault-tolerant composition of mobile agent systems.

The internal agent failure – a failure from the third category – should be detected inside an agent. All the traditional recovery techniques developed for sequential programming can be used to deal with such failures. If an agent fails to recover from a failure individually, then there is a need for cooperative exception handling by all the involved agents.

The last category of failures corresponds to exceptional situations in the environment that are beyond the control of a mobile agent. Examples of this are failures of hardware components, administrative restrictions, software bugs in the underlying middleware and in the core components of the environment. All these failures are typical of the domain of mobile agent software and mobile agents usually cannot anticipate or avoid this kind of malfunctions.

In RODIN, we created a methodology supporting systematic integration of the fault tolerance mechanisms into development of CAMA applications. We have demonstrated how to specify the fault tolerance mechanisms formally as an intrinsic part of the system specification. For instance, an application of the proposed approach in the Ambient Campus case study allowed us to develop fault tolerant middleware capable to tolerate agent disconnections.

In parallel with our work on the top-down approaches to the development of pervasive systems, we have put forward bottom-up, model-checking approaches. We have identified the typical behavioural patterns of pervasive systems and proposed model-checking techniques optimised for these patterns. This strand of research enabled verification of temporal properties of *configurations* – sets of cooperating agents. In the proposed approach, the agent system evolution is constrained by a set of properties. A process algebra notation is used to describe configurations and then the Petri net-based model checker is applied for verification. The range of verified properties includes deadlock freeness, proper use of the scoping mechanism and other application specific properties. The formal top-down development provides a valuable input to model checking. For instance, complex invariant properties derived in a refinement chain are used to verify that the agents are forbidden to execute illegal operations in the scope.

As a case study, we have developed an Ambient Campus – a multi-agent application facilitating studies of students via mobile hand-held devices. The system runs on the CAMA middleware distributed over

several students' devices such as PDAs, smartphones, laptops, and a lecturer's desktop computer. In the lecture scenario, the lecturer is able to register students when the lecture starts, create groups and assign students to these groups, monitor the group work and assist groups, when necessary. Another implemented scenario is developed to help new students arriving at a university campus with the registration process. The system consists of a traditional university campus, virtual campus and ambients. Automated student registration complements manual registration process, so that in case of failure it switches to the completely manual mode. The system helps the new students to get acquainted with the campus facilities and help them to navigate through various student activities.

The work described above has contributed not only to creating a methodology for developing resilient pervasive systems but also to building the RODIN tool platform [14]. By exercising the platform in modelling multi-agent systems, we enhance its capability in modelling and verifying novel complex computational phenomena, such as resilient pervasive systems.

5. From RODIN to DEPLOY

We have just started a new EC project on Industrial Deployment of System Engineering Methods Providing High Dependability and Productivity – DEPLOY [9]. DEPLOY aims to make major advances in engineering methods for dependable systems through the deployment of formal engineering methods. In this project we will continue our work on ensuring resilience of complex pervasive systems.

Our focus is on developing patterns supporting modelling and refinement that assist system developers in smooth and systematic integration of resilience into system development. Within DEPLOY, we are planning to extend our approach to modelling resilient pervasive systems by proposing new formal patterns [8] supporting introduction of different fault tolerance and communication mechanisms into the rigorous development process.

6. Acknowledgments

This work is supported by the FP7 DEPLOY project. A. Iliasov is partially supported by the EPRSC/UK TrAmS project.

7. References

[1] J.-R. Abrial. The B-Book. Cambridge University Press, 1996.

[2] J.-R. Abrial. Event Driven Sequential Program Construction. www.matisse.qinetiq.com, 2000.

[3] J.-R. Abrial, D. Cansell, D. Mery. Refinement and Reachability in Event B. In Proc. ZB'2005: Formal Specification and Development in Z and B. LNCS 3455, 2005, pp. 222-241.

[4] M. Butler, E. Sekerinski, K. Sere. An Action System Approach to the Steam Boiler Problem. In Formal Methods for Industrial Applications. J.-R. Abrial, E. Börger, H. Langmaack, Eds. LNCS 1165. 1996.

[5] G. Di Marzo Serugendo, J. Fitzgerald, A. Romanovsky, N. Guelfi. A metadata-based architectural model for dynamically resilient systems. In Proc. the 2007 ACM Symposium on Applied Computing (SAC), Seoul, Korea, March 11-15, 2007. ACM, 2007, pp. 566-572.

[6] A. Iliasov, A. Romanovsky. CAMA: Structured Coordination Space and Exception Propagation Mechanism for Mobile Agents. ECOOP 2005 Workshop on Exception Handling in Object Oriented Systems. Glasgow, UK.

[7] A. Iliasov, A. Romanovsky, B. Arief, L. Laibinis, and E. Troubitsyna. A Framework for Open Distributed System Design. In Proceedings of Computer Software & Applications Conference (COMPSAC 07), Volume II, pp. 658-668, July 2007, Beijing, China. IEEE CS, 2007.

[8] A. Iliasov, A. Romanovsky. Refinement Patterns for Fault Tolerant Systems. In Proc. the 7th European Dependable Computing Conference (EDCC-7). Lithuania, May 7-9, 2008.

[9] Industrial Deployment of System Engineering Methods Providing High Dependability and Productivity - DEPLOY, FP7 project. deploy-project.eu

[10] C.B. Jones, I.J. Hayes, M.A. Jackson. Deriving Specifications for Systems That are Connected to the Physical World. Formal Methods and Hybrid Real-Time Systems: Essays in Honour of D. Bjorner and Z. Chaochen, LNCS 4700, pp. 364-390, 2007.

[11] L. Laibinis, E. Troubitsyna, A. Iliasov, A. Romanovsky: Rigorous Development of Fault-Tolerant Agent Systems. Rigorous Development of Complex Fault-Tolerant Systems. LNCS 4157, 2006, pp. 241-260.

[12] Rigorous Open Development Environment for Complex Systems (RODIN), IST FP6 project. http://rodin.cs.ncl.ac.uk/

[13] Rigorous Open Development Environment for Complex Systems (RODIN). Deliverable D7, Event B Language, http://rodin.cs.ncl.ac.uk/

[14] The RODIN platform. rodin-b-sharp.sourceforge.net/

FADA: Formalisms and Algorithms for Resilient Services Design in Ambient Systems[‡]

Matthieu Roy
Marc-Olivier Killijian
LAAS-CNRS
Université de Toulouse
Toulouse, France
roy | mkilliji@laas.fr

Sara Tucci Piergiovanni
Leonardo Querzoni
Silvia Bonomi, Sirio Scipioni
University of Roma La Sapienza
Roma, Italy
querzoni @dis.uniroma1.it

François Bonnet

IRISA
Campus de Beaulieu
Rennes, France
fbonnet@irisa.fr

Abstract

In this paper, we describe the aims and preliminary results of FADA, a framework for developing resilient services and reasoning on mobile systems.

1. Introduction and objectives

The advent of massively distributed systems in which every user carries an entity with full computing power, communication, storage and positioning capabilities, allows us to envision many new applications and services, truly decentralized by nature and tightly coupled to the position of entities. Nevertheless, the deployment of such systems imposes the definition of formalisms that capture the new features of these systems and allow algorithms, mechanisms and architectures to be developed and reasoned about.

Indeed, for the first time ever, there exists a strong coupling between the graph of possible interactions between entities and the geographical distribution of nodes. The strength of the classical Internet model was to abstract all communication links into a complete graph, where any two nodes willing to cooperate could open a connection. This clique-based system modeling can no longer be applied to distributed dynamic systems where entities can communicate using short-range wireless technologies.

As pointed out by the ReSIST NoE research agenda [1], two main issues arise when trying to deal with such large-scale and ambient systems:

- The evolvable nature of mobile systems imposes any mechanism built for them to be resilient to mobility- and failure-induced changes to the composition and/or topology of the system.
- New features of such systems are not represented in traditional distributed models, namely their dynamicity and locality, and more generally the geographical distribution of entities.

The overall objective of FADA is to propose formalisms, algorithms and architectural patterns towards solving those issues.

2. Scientific approach

In this work, we address the problem of building resilient services on a distributed, mobile system in a cooperative way. In order to have a complete and sound solution, our approach focuses on the following aspects of the problem:

- Formalization of the system into a model that goes beyond traditional distributed systems models, by including mobility and geographical information, and by adapting failure models,
- Architectural design definition of the basic building blocks (i.e., abstractions) for cooperative services implementation, based on the above model,
- Development of algorithms to implement cooperative resilient services, despite limited knowledge, arbitrary size, mobility patterns and an unbounded number of failures,
- Assessment of the proposed algorithms using simulators, and then on experimental platforms.

In this work, we follow a horizontal and a vertical approach. Horizontally, we started to design an architectural landscape for cooperative services on top of a system composed of mobile nodes, by identifying basic building blocks, providing relative abstractions and implementation algorithms. As presented below, we focused on the traditional abstractions of distributed systems, namely coordination, synchronization, agreement and reliable communication. Vertically we will focus on the problem of providing a reliable, geographically localized blackboard system, modeled as a reliable storage service.

3. Research goals

To address the above problems, the first step consists in identifying a suitable formalization of the system that takes mobility into account. In this work, two major mobility modes are considered:

- *Active* mobility: in this case, entities can move on demand or by themselves, and thus movements can be part

[‡] This work is fully funded by the ReSIST Network of Excellence (FP6 IST contract 026764)

of the solution. As an example, a system composed of mobile cooperating robots can be reconfigured spatially to solve a particular task.

- *Passive* mobility: entities move independently of the problem to be solved. For example, this mobility model represents a system where users carry a device (phone, computer); the path followed by a device is bound to the user's movements and the algorithms implemented on the device must provide the service despite various movement patterns.

From the theoretical point of view, the aim of this work is to show that using a formal model of the system allows the definition of conditions on mobility patterns and node density to be able to implement basic resilient services that can be formally proven correct.

From the practical point of view, we aim at providing the definition of an architecture for resilient services in mobile systems. Moreover, the basic building blocks provided within this architecture will be evaluated both on simulators and on reduced-size real systems, thus improving confidence in the pertinence of the formal system models and algorithms.

4. Architectural description

Reasoning on constantly evolving systems imposes the need for suitable abstractions that capture the very nature of interactions between entities. Following the seminal work of Chockler et al. [2], we architectured our model using three layers, as depicted in Figure 1. :

- The lower layer is composed of abstractions that are close to the hardware. [2] identified three abstractions for mobile systems:
  - A *timed local broadcast* service that sends messages in at most ∂ time units but can lose messages. This service refers to hardware properties of the communication medium.
  - A *collision detector* is an oracle which definition is close to classical failure detectors [3]. It encapsulates the additional formal assumptions needed to provide reliable algorithms such as consensus or reliable broadcasts.
  - A *wake-up* service determines which node can send message to prevent collisions between messages.
  - Additionally, since we are interested in the interactions of users with the physical world, we propose a fourth abstraction, the *Localization and Clock* service, that provides information available from a GPS-like device, i.e. localization information, and a global clock service.
- The intermediate level provides higher-level services that integrate physical (localization-based) and logical (network-based) information:
  - The *proximity map* exports a hybrid map of an entity's neighborhood that indicates the position of nodes that are within communication range.
  - The *node failure/leave/join detection* service is meant to detect changes in the configuration of the distributed system.
  - The *time fair-loss local broadcast* is built using timed local broadcast and wake-up service: each message suffers from a maximal delay ∂, but messages are not systematically lost.
  - The *quiescent asynchronous reliable broadcast* service[4] is a broadcast service that ensures that messages are not sent forever, i.e. that retransmissions occur only a finite number of times.
- At the higher level, one can find simple reliable abstractions, such as geo-localized atomic registers, test&set operations, or localized consensus.

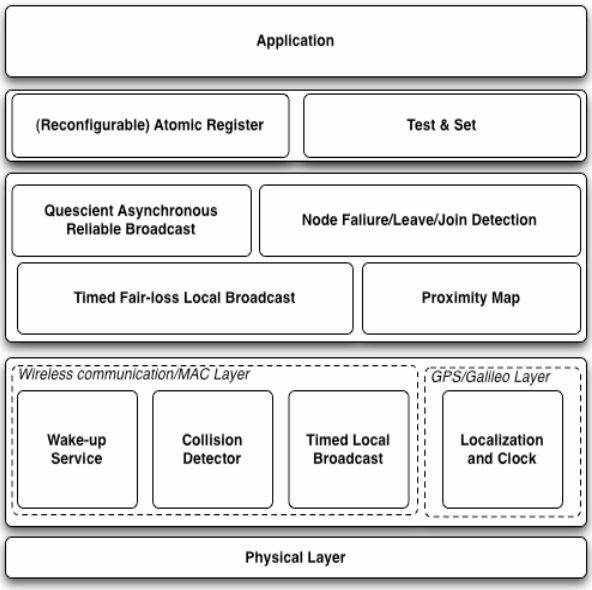


Figure 1. FADA architectural description

Current and further work includes formalizing the assumptions of the above building blocks, specifying and developing the middleware associated to them. Evaluation of the algorithmic solutions will be based on our proof-of-concept application, namely a localized blackboard system.

5. References

[1] ReSIST (Resilience for Survivability in IST) website: http://www.resist-noe.org

[2] Chockler, G., Demirbas, M., Gilbert, S., Newport, C. and Nolte, T. : *Consensus and Collision Detectors in Wireless Ad Hoc Networks*. ACM PODC 2005.

[3] Chandra, T. D. and Toueg, S. : Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, *43*(2), 225-267 (1996).

[4] Bonnet, F., Ezhilchelvan, P. and Vollset, E. : *Quiescent Consensus in Mobile Ad-hoc Networks using Eventually Storage-Free Broadcasts*. ACM SAC 2006

The FAERUS Project: Formal Analysis of Evolving Resilient Usable Systems

G. Faconti[1], M. Harrison[2], M. Massink[1], P. Palanque[3]
(1) CNR-ISTI, Pisa, IT (2) Newcastle University, UK (3) IRIT, Toulouse, FR
G.Faconti@isti.cnr.it Michael.Harrison@newcastle.ac.uk M.Massink@isti.cnr.it
palanque@irit.fr

Abstract

This paper gives a brief overview of the research taking place in the context of the FAERUS project which is one of the mini-projects affiliated to the EU-IST Network of Excellence RESIST (IST-2006-026764).

1. Goals and topics

The proposed research is based on the vision of a future of pervasive networked devices, able to interface with the environment to support context-aware services; diverse interaction techniques through dynamic reconfiguration of devices made available to the users depending on context situations; large-scale, dynamic, disconnected networks providing a huge variety of services.

All models of computing, based on these premises, share a vision of small, inexpensive, robust networked processing devices, distributed at all scales throughout everyday life and generally turned to distinctly daily ends. However, contemporary human-computer interaction models are inappropriate and inadequate to the case. This suggests that the "natural" interaction paradigm appropriate to an ubiquitous computing environment has yet to emerge though it is recognised that in many ways we are already living in a ubiquitous computing world. The design methodologies and tools for ubiquitous computing and communication systems need a shift in focus beyond the conventional engineering approach of ``predictability'': from performance/robustness to resilience, where evolve-ability is a constituent system property.

With this view, users and human operators of ubiquitous systems are increasingly exposed to continuously changing environments and contexts and to a diversity of interaction devices. This situation raises important issues about the usability of systems where users are traditionally considered as a source of errors because of their inherent unpredictability. However, recent studies show that human behaviour is not completely non-deterministic but in most cases follows certain probability distributions. In addition, lack of usability (e.g. ambiguous presentations) has been proved to be an important source of errors and mistakes.

This project aims to investigate usability issues in ubiquitous systems by taking a User Centred Approach to system design with a focus on gaps modelling non-functional requirements and diversity of interaction techniques, and to a lesser extend also on context confusion.

The project has two main objectives. The first objective is to study probabilistic models for the dynamic reconfiguration of multi-modal interfaces preserving a minimal basic set of usability requirements. IRIT has already developed models for handling the functional aspects of multi-modal interactions including speech, mice and touch for an interactive cockpit application in new commercial aircraft. In our research we plan to extend these models with quantitative probabilistic aspects in order to be able to assess the performance aspects of such multi-modal interaction techniques. This serves to gain experience in modelling dynamically re-configurable interfaces in well-studied environments in order to assess the feasibility of applying the approach to interfaces in pervasive systems.

The second objective of the project is to investigate recent modelling and verification techniques that allow for the analysis of the effect that the behaviour of a large number of users may have on the functioning of the system. In this respect, we plan to study a concrete case-study, namely a system supporting collaborative work such as those often used in the manufacturing industry, where designers access and modify CAD designs of products. The various phases in such design, ranging from initial development of product part designs, to close-to-the-deadline frequent small adjustments of CAD designs, leads to rather different use-patterns of the same system. This may potentially lead to considerable changes in responsiveness of the system, and therefore influence usability. Models that

could inform designers of those systems on such effects would be very useful and provide another way to integrate user and usability aspects in the overall system design.
These objectives are linked together by a common objective. In understanding what are quantitative characteristics of human behaviour in relation to interaction with technology it is normal to describe this behaviour either in terms of a task model (what it is observed that the user typically does to achieve goals) or a user model (capturing relevant cognitive and motor characteristics). An important problem here is to provide a task or user model that is abstract enough to capture user behaviour accurately (the more detailed the more likely to be diverted from) while at the same time detailed enough to provide formative information about the effectiveness of the design. The two examples considered in this project are concerned with quantitative aspects at two levels. The multimodal example requires a detailed model to provide understanding of the particular characteristics of the different modalities while the second example requires very abstract and simplifying models capturing the essence of multiple concurrent users.
The project will help understanding these different levels of modelling more effectively and assessing the feasibility of defining the probabilistic models of interaction behaviour relevant to the level of analysis.

From another point of view, smart environments could be developed that inform users about the current situation and guide them to adopt behaviours that help to avoid possible congestion of the system, or at least, to reach a situation with better performance properties for what concerns usability. Such techniques could be applied in many cases where users share a common space, whether collaborative systems and virtual spaces or physical spaces such as in airports, on highways.

2. Scientific Approach

The scientific approach followed in this research is that of the application of formal modelling and analysis approaches such as process algebras and Petri Nets, to model the Human-Computer Interface aspects of the above described situations. In particular, recently, stochastic and probabilistic extensions of these approaches have been developed together with several tools for their formal verification such as probabilistic and stochastic model-checkers. In the proposed study we will investigate the feasibility of the use of these techniques for the analysis of non-functional aspects of multi-modal interfaces and group-ware applications.

One of the complications that usually arise when modelling systems with many similar elements, e.g. like many users using many different but similar resources such as in a collaborative system, or many users of a common virtual or real space activating sensors and reacting to provided information such as in airports, is the sheer number of elements to model, which usually leads to unmanageable state-spaces. So, a further technique that we intend to investigate is the so called Fluid Flow Approximation that works with a continuous abstraction of discrete numbers and is based on the solution of sets of ordinary differential equations. We expect such a technique to provide information on the performance of the system under particular usage patterns. For example, how does the time to obtain a file change when many users of the collaborative system make frequently small modifications to files compared to infrequent but time-consuming modifications. Or, in an airport evacuation we might like to be able to address questions like: if some people interpret provided information in a wrong way and take a wrong route, what is the probability that congestion (and probably panic) will arise and where?

All of the above techniques are quite new and not much experience has yet been gained with their application to HCI-related problems. In this respect, the involved partners provide complementary expertise to the project. In fact, IRIT has much experience with the analysis of multi-modal interaction in safety-critical systems and Pisa/CNR-ISTI contributes with its expertise in probabilistic and stochastic model-checking and other forms of formal analysis in the context of human-computer interaction. The University of Newcastle has special expertise in the analysis of human interface problems in ubiquitous systems which is of utmost relevance to the project.

3. Project Organisation

Project leader: Mieke Massink (CNR-ISTI)

Project Participants: Mieke Massink (CNR-ISTI), Maurice ter Beek (CNR-ISTI), Jean Francois Ladry (IRIT), Marco Winckler (IRIT), Jeremy Bryans (Univ. of Newcastle)

Senior advisers: Giorgio Faconti (CNR-ISTI), Philippe Palanque (IRIT) , Michael Harrison (Univ. of Newcastle).

4. Selected References

[1] M.H. ter Beek, M. Massink, and D. Latella – “Towards Model Checking Stochastic Aspects of the 'thinkteam' User Interface.” In Interactive Systems: Design, Specification, and Verification - Revised papers of the 12th International

Workshop on Design, Specification, and Verification of Interactive Systems (DSVIS'05), Newcastle upon Tyne, UK (S.W. Gilroy and M.D. Harrison, eds.), Lecture Notes in Computer Science 3941, Springer, Berlin, 2006

[2] M.H. ter Beek, M. Massink, D. Latella, S. Gnesi, A. Forghieri, and M. Sebastianis, "A Case Study on the Automated Verification of Groupware Protocols". In Proceedings of the 27th International Conference on Software Engineering (ICSE'05), St. Louis, MO, U.S.A., ACM Press, New York, 2005.

[3] G. Doherty, M. Massink, and G. P. Faconti. "Reasoning about interactive systems with stochastic models". In C. Johnson, editor, Interactive Systems: Design, Specification, and Verification, volume 2220 of Lecture Notes in Computer Science. Springer-Verlag, 2001.

[4] Harrison, M.D., Kray and Campos, J.C. (in press) "Exploring an option space to engineer a ubiquitous computing system". In P. Curzon and A. Cerone eds. Proceedings of Formal Methods in Interactive Systems 2007. Electronic Notes in Theoretical Computer Science, 2007.

[5] Harrison, M.D., Kray, Zhiyu Sun and Huqiu Zhang. "Factoring user experience into the design of ambient and mobile systems". EIS'07, 2007.

[6] J. Hillston, "Fluid Flow Approximation of PEPA models". QEST'05, IEEE, 2005.

[7] D. Navarre, P. Palanque, J.-F. Ladry. "Model-based Framework for Addressing Diversity, Usability and Reliability for Safety Critical Interactive Systems". In : International Conference of the IEEE Industrial Electronics Society and Conservatoire des Arts et Metiers (IECON 2007), London, UK, 22-24/10/07, IEEE, 2007.

A dependable Wireless Sensor Network for early wildfire detection

S. Blanc, P. Yuste, A. Bonastre
Department of Computer Engineering, Technical University of Valencia, Spain
{sablacla, pyuste, bonastre}@disca.upv.es

Abstract

This paper describes an on-going work about a dependable Wireless Sensor Network (WSN). The WSN is oriented to forest wildfire risk evaluation, fire detection and evolution monitoring in large outdoor areas.

1. Introduction

Wild forest fire prevention, detection and tracing requires dependable computational systems. Parameters such as temperature, humidity, wind force, barometric pressure, etc., are measured and computed throughout big forest extensions. Thus, a wireless ad-hoc connection is conceptually the best option suitable to support the application.

2. Dependability aspects

A wildfire prevention, detection and trace prediction application is been built on a Wireless Sensor Network (WSN) [1]. Dependability aspects are:
Coverage: The number and distribution of sensors as well as the computational system used to transmit information should cover the biggest number of different points inside the area providing a real time useful monitoring system.
Detection latency: The time from a sensor detection until its reception in a monitoring interface is especially critical [2]. A dependable system should guarantee a maximum constant detection latency time below the maximum required by the application.
Fault tolerance: To achieve a full surveillance system it is needed to avoid areas without sensor measurement ("holes") for either missing or out of service nodes. There are diverse "holes" causes, including human aggressions, low energy or natural accidents. Apart from making nodes more resilient, the system needs a reliable network protocol able to avoid routing data through "holes" and grant a complete network connection without isolated nodes. Minimum detection latency of missing or out of service nodes is required.
Quality of Information: While some messages in the network need high quality of data (i.e. alarms), other messages are only used to send non-critical measurement data or even perform protocol tasks. A well balanced protocol can assume some message loss while assuring alarm-messages delivery with a maximum known latency time.
False alarms: Reducing false alarms implies three issues to work on: making resilient nodes, establishing optimal sensor parameters, and obtaining sensor-detection redundancy (more than *n* nodes detect the same possible fire risk).
Security attacks: Communication protocols must grant confidentiality, integrity and authenticity. Most wildfires are intentionally provoked [2], so the system must be able to detect and reject external assaults, and even avoid to be tampered.
Energy efficiency: Apart from protocol energy saving strategies, nodes must be built to have a long life and provided with rechargable energy supplies.

3. Sensor distribution

Sensors are part of something more complex: the **node** that is also able to compute and transmit data and even acts as a communications infrastructure for other nodes. The distribution of nodes in the forest is based on area subdivision. Each sub area is monitored by one node. The sub area size must be defined between an optimal and a maximum size. The knowledge of the geographical conditions of the area helps to define the subdivision. For example, temperature or humidity parameters have a very low or null variation in short distances, unless natural exceptions such as drops of importance, vegetation thickness variations, etc.

Moreover, there are other aspects used to set both limits. As size increases, false alarms from directional sensors (for example, infrared) also increase, as well as the detection latency time in case the fire starts in a place far from the sensor. As size decreases, the

application gets better connectivity, sensor coverage and redundancy on the detection, allowing the collaboration between several sensors to confirm the fire; however the traffic of messages and the cost of each node must be considered. Components, such as GPS, infrared sensors, solar panels to provide power, etc., increase the node cost and make them more valuable and desirable by thieves. Besides, the maintenance of very dense WSN networks in large outdoor areas implies a considerable cost in human resources. An additional aspect to be considered is the environmental impact of electronic instrumentation in our forests. Degradation of nodes (or no degradation) and animal-life risks should always be considered.

In the proposed WSN, it is expected that most time sub areas are monitored in their optimal size, while they will enlarge till a maximum to obtain redundant data information or cover "holes" left by disconnected or moved nodes. "Holes" are covered by active neighbour nodes. Neighbourhood depends on distance, geographical distribution and power level. The objective is to grant a fire-detection latency time below the maximum required by the application in optimal conditions as well as in the presence of faults.

4. Network topology

The proposed WSN is based in a sink scheme with clustering. Nodes disconnect to the network due to failures. The early detection of a disconnection is important for an efficient covering of the "hole". Moreover, if a node fails but it doesn't become silent, errors can propagate through the network, even ending up collapsing the net (for example, in case of a babbling idiot transmission).

The WSN proposes a clustering structure, usually used for power control [3], [4], but in this paper proposed for fault tolerance. The goal is to gain a minimum detection latency of disconnected or failed nodes. The clustering is based in TDMA transmission, where all the nodes in the cluster must transmit one message per TDMA. Messages will be either "alive" messages or "data" messages depending on if the node has measured data to send or not.

In a clustering structure, there are two types of transmission: within cluster and between clusters. The message transmission in our clusters is used to: maintain the synchronism of the cluster nodes, detect failures and perform data fusion tasks. On the other hand, messages transmitted between clusters are always "data" messages, where data include both sensors information and cluster incidences such as shut downs and failures. Thus, the message transmission between our clusters is used to: detect a whole cluster shut down (no one in the cluster is alive) and transmit information obtained by both sensors and fault tolerance mechanisms.

The Cluster Heads are dynamically selected based on power level and geographical location. Cluster Heads transmission to the sink is assumed as multi-hop. The routing scheme implies some location techniques. Location will be deducted with GPS or without it by triangulation algorithms which permit each node to know its current position on the map. Knowing the geographical location of nodes simplifies network routing and clustering formation, but moreover it is important for monitoring tasks. For example, in a sink scheme, a real time interface that uses the sink information helps to human users for a quick reaction in case of receiving an alarm.

Finally, nodes are provided with mini- solar cells. However, nodes have to enter in "sleep mode" from time to time to save energy. A slept node is a disconnected node to the network (it can be considered as a special type of failure). Then, its "hole" needs to be covered by active neighbour nodes. Difficulty becomes in avoiding more than *i* neighbour nodes in sleep mode. The proposed WSN algorithm will define a maximum known *i*.

5. Current work

Currently, a first version of the network protocol has been developed and its dependability properties are being evaluated. The feedback obtained will help in defining improvements. Also, the currently implemented prototype covers a medium sensor network. Protocol specification should be adapted for covering the dependability aspects of larger sensor networks.

6. References

[1] F. L. Lewis, Wíreless Sensor Networks, Smart Environments: Technologies, Protocols, and Applications, John Wiley, New York (USA), 2004.

[2] F. Romero, F. Serna, Grandes Incendios forestales: Causas y efectos de una ineficaz gestión del territorio, WWF/Adena, Madrid (Spain), June 2006.

[3] M. Ye, C. Li, G. Chen, J. Wu, EECS: An Energy Efficient Clustering Scheme in WSN, IPCCC 2005. IEEE. April 2005.

[4] W. Heinzelman, A. Chandrakasan, H. Balakrishnan, Energy-Efficient Communication Protocol for Wireless Microsensor Networks, Hawaii IC on System Sciences, vol. 8, 2000.

Toward Adaptable Software Architecture for Dependability: ASAP

Thomas Robert and Thomas Pareaud
Université de Toulouse
LAAS-CNRS
7 avenue du colonel Roche,
31400 Toulouse
France
{trobert, tpareaud}@laas.fr

Inga Zutautaite-Seputiene
Vytautas Magnus University
Mathematics and Statistics Department
8 Viliekos str.,
LT-44404 Kaunas
Lithuania
ingaz@isag.lei.lt

Vladimir Stankovic
City University,
Centre for Software Reliability,
London EC1V 0HB,
United Kingdom
v.stankovic@city.ac.uk

Shen Lin
Lancaster University,
Computing Department
Lancaster LA1 4WA,
United Kindgom
s.lin6@lancaster.ac.uk

Abstract

This short paper describes the ASAP mini-project (Assessment-based Adaptable Software Architecture for Dependability) that was launched within the Network of Excellence ReSIST[1]. The main objective of the work is to address the adaptation of the fault tolerance software in a system according to on-line assessment-based triggers. On-line assessment and adaptive fault-tolerance have been identified in ReSIST as major research gaps for the future. The ambition of the mini-project is to gather recent work on fault tolerance software adaptation using open-component software based engineering techniques and advanced knowledge on on-line assessment using Bayesian inference. Some small case studies are planned to perform and early validation of the ideas and proposed approach.

Keywords

On-line assessment, adaptive fault tolerance, Bayesian inference, reflective computing.

1. Introduction

The evolution of systems at runtime is currently to be addressed in many application domains, those being critical or not. A solution to cope with this requirement for more flexible systems is to adapt the system design and service at run-time. The on-line adaptation may be required for many reasons like changes in the environment, components upgrades, system configuration, resource consumption, components transient failure but also new fault assumptions. These are some of the possible triggers for adaptation of the system. This kind of problem is concern safety critical embedded systems with limited resources and no means to repair, ubiquitous computing or mobile systems, and also large-scale distributed systems.

The solution must tackle two kinds of problems:

- The assessment of operational conditions and detection of a change to trigger the adaptation
- The adaptation of the target system, including in particular its mechanisms of achieving high dependability (e.g. fault tolerance).

The objective of this work is to propose an architecture, and methods to make adaptive resilient systems, in particular regarding fault tolerance. To reach this aim, we introduce several assessment techniques and a reflective component model as base modules to build up such an architecture. The reflective model is drawn from a complete open-component engineering framework.

This work is the continuation of the analysis of research challenges performed within the framework of the network of excellence ReSIST, in [3]. Among them, **Adaptation and self-organisation** concerns resilience of an evolving system that is highly affected by its ability to adapt to new requirements of the

[1] Network of Excellence ReSIST (*Resilience for Survivability in IST*, http://www.resist-noe.org)

environment. The problem is thus to dynamically reorganize the system, including its resilience mechanisms, according to new operational conditions.

In the context of fault tolerance adaptation, the two problems identified above have been spelled out more precisely as:

- An on-line assessment engine that deals explicitly with the assessment at runtime of the system resilience. Thus, the on-line (run-time) assessment is concerned with system operating in highly dynamic and evolving environment (e.g. network congestion, new/changing service customers, effects of exploits of faults / vulnerabilities). Its goal is assessment of the implications of the changing environment on system resilience ranging from simply signaling a sub-standard operation to triggering a system's adaptation.
- A design for adaptation applicable to fault tolerance –identified as an important research gap in [3]. It has been pointed out that the adaptation capabilities of a system depend on the early stages of its design, and the software development technology that was selected. In the case of fault tolerance, they identified the urging need for an environment that offers system architects the means to describe and design the adaptation of the fault tolerance [8].

This paper gives the scheme of an architecture to provide on-line adaptation of resilient systems. Such an architecture is built from different approaches and techniques to address the sub-tasks identified above.

2. Overall architecture

The system architecture we envisage will consist of several abstraction layers as illustrated in Figure 1 that gives a coarse view of the system. The lower layer corresponds to the system services provided to the user and defined in the functional specification documents. The fault tolerance layer corresponds to the set (pool) of possible mechanisms required to fulfill non-functional specifications (give examples?). The mechanisms may require different degrees of redundancy (including diverse redundancy) and offer different degrees of dependability assurance. The adaptation layer corresponds to the various scenarios enabling the fault tolerance strategies to be changed on-line. Last, the assessment layer corresponds to the set of mechanisms responsible for estimating the current situation and deciding which of the available fault-tolerant mechanisms is the most adequate for it. If the currently deployed mechanism is sub-optimal, then a change will be triggered. This change will lead to modifications of the active fault-tolerance mechanism and, if necessary, modifications of the base system. In practical terms, these layers correspond to components such as Base System (BSys), Fault Tolerance mechanisms (FTM), Adaptation Engine (AdE), Assessment Engine (AsE) that uses Failure Detectors (FDs), Software Sensors (SWS) and Software Actuators (SWA).

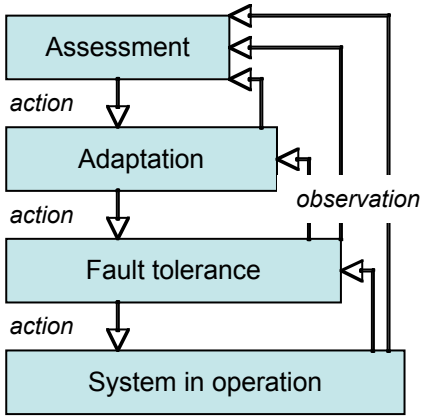


Figure 1. Abstraction layers for on-line adaptation

Figure 1 depicts in fact the proposed reflective architecture, each abstraction level corresponding to a metalevel. In reflective terms, observation and control facilities provided by open component technologies enable adaptation to be performed on each software level configuration on-line.

The **Base System** is the application software (and its supporting executive layers, operating system and middleware) without any fault-tolerant mechanisms, e.g. embedded software for automotive application, avionics, etc. Such subsystem consists of several components, with their respective APIs, which allow for interaction[2].

The **Fault Tolerance** software is defined as a set of interacting components that provides fault tolerance according to some requirement and resources availability. Their organization must be flexible enough to be manipulated at runtime. Novel software engineering techniques such as Fractal [1] or OpenCOM [2] can be used to reach this aim.

The **Adaptation** corresponds to a set of programs that are able to change the configuration of the fault tolerance software in order to modify on-line a given fault tolerance strategy. Because the open component technology enables individual components of a software layer to be manipulated at runtime (dynamic

[2] In case diverse components are used to achieve tolerance against software faults, the diverse components are still seen as part of the Base System, since they are assumed to be available (e.g. off-the-shelf) and developed by third parties in accordance with the 'design for adaptation' principles.

bindings), their objective is to update a given strategy while maintaining consistency.

The **Assessment** is a process of measuring the non-functional characteristics of the system, using a model of the system and the evidence available before and during operation. The Assessment in this particular framework is achieved by an Assessment engine, a piece of software, which allows for all necessary elements of the assessment to be represented: the system model, the pre-deployment knowledge about the non-functional property of interest, and the mechanism of updating the knowledge about the attribute of interest when new evidence becomes available. For the sake of concreteness in this report we limit ourselves to the following examples of assessment:

- Computation of dependability measures of interest associated with the particular deployment of BSys, AdE and FDs , e.g. the probability of system failure on demand, using Bayesian inference as described in a series of papers [4], [5]. Such a service ensures an up to date estimation of the dependability level of the whole system.
- Detection of the changes in the operational use of the system, i.e. of its operational environment, e.g. as a result of changes in the pattern the software is being used, which may invalidate the dependability measures computed in the past under different operational conditions.
- Detection by anticipation of real-time failure due to changes in the runtime conditions (i.e. scheduling errors) that lead to abnormal behaviours or erroneous states [6]; the approach consists of analyzing correct continuations using the automata time abstraction [7].

Finally, **SoftWare Sensors** will monitor the work of the deployed components and deliver input data to the specific Assessment engine deployed to keep the assessment up to date. The access to SWS will be via a defined API, in fact a meta-interface providing inputs to the adaptation and assessment engines.

Similarly, the adaptation will need means to act upon the system in operation through **SoftWare Actuators**.

3. Conclusion

Clearly, the on-line adaptation may be required for multiple reasons like changes in the environment, components upgrades, system re-configuration, resource usage, components transient failures as well as new fault assumptions. These are some of the possible triggers for adaptation. These situations are quite generic and apply to a wide range of systems, from resource-constrained safety critical embedded systems that cannot be repaired on-line, ubiquitous computing and mobile systems, to large-scale distributed infrastructure.

We argue that any solution to these challenges needs to address the following two problems:

1) How to properly assess operational conditions and trigger adaptation when required

2) Once adaptation has been decided, how to adapt a target system, and in particular how to adapt its dependability mechanisms.

In practice, the system in operation is composed of services implementing the functional specifications, those services being run on top of a middleware composed of an *assessment engine*, an *adaptation engine* and *fault tolerance mechanisms*.

The assessment engine obtains inputs from the system and its environment, and uses them to compute adaptation trigger. These triggers are passed on to the adaptation engine that is then responsible for dynamically modifying the software configuration (in particular the fault tolerance strategies) according to the needs that have been identified.

4. Acknowledgments

The authors want to give special thanks to Jean-Charles Fabre (*LAAS-CNRS, Toulouse*) and Peter Popov (*City University, London*) for their help in the elaboration of this mini project. They both played an important role in the organization of the project as well as in the contribution to the project content. We also thank François Taïani (*Lancaster University)* for his participation to this project.

5. References

[1] Bruneton, E., et al., *The Fractal Component Model and Its Support in Java*. Software Practice and Experience, 2006. 36 (11-12)(Experiences with Auto-adaptive and Reconfigurable Systems): p. 29.

[2] Coulson, G., et al. *Towards a Component-based Middleware Architecture for Flexible and Reconfigurable Grid Computing*. in *Workshop on Emerging Technologies for Next generation Grid (ETNGRID-2004), 13th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises*. 2004. Italy.

[3] M. Banatre (Editor), *From Resilience-Building to Resilience-Scaling Technologies: Directions – Chapter 1, Evolvability of Resilient Systems*, Report D13, RESIST NoE, N°026764, Resilience for Survivability in IST, September 2007, 130 pages.

[4] Littlewood, B., P. Popov, and L. Strigini. *Assessment of the Reliability of Fault-Tolerant Software: a Bayesian Approach*. in *19th International Conference on Computer Safety, Reliability and Security, SAFECOMP'2000*. 2000. Rotterdam, the Netherlands: Springer.

[5] Popov, P. *Reliability Assessment of Legacy Safety-Critical Systems Upgraded with Off-the-Shelf Components*. in *SAFECOMP'2002*. 2002. Catania, Italy: Springer-Verlag.

[6] Robert, T., M. Roy, and J.C. Fabre, *Real-time run-time verifiers: theory and practice*, LAAS Research Report 07276, 2007, 10p.

[7] Alur, R. and D.L. Dill, *A theory of timed automata*. Theoretical Computer Science, 1994. 126: p. 183--235.

[8] T. Pareaud, J-C. Fabre, M-O. Killijian, Towards adaptive fault tolerance in Embedded Systems, International Congress on Embedded Real-Time Software, Toulouse, France, Jan. 2008, 9p.

Estimating Fault Tolerance Overhead with Self-Scaling Benchmarks

Ronald J. Leach
Howard University
rjl@scs.howard.edu

Abstract

Run-time overhead of fault tolerance techniques can be estimated using experiments to determine the efficiency of interprocess communication (ipc) and network communication techniques. We describe an approach in which simple benchmark code can be instrumented in order to provide assessment of overhead. The method can be used to good effect with self-scaling benchmarks. This approach to measurement of ipc and network communication efficiency is likely to be especially important to the fault tolerance research community as multi-core processors and virtual servers, such as those that are becoming available using VMware become more prevalent.

1. Introduction

Nearly every technique used for fault tolerance has a run-time overhead affecting performance. There may be a general, predictable degradation in performance due to checking correctness during system operation. There may also be a rapid change in performance if a fault is detected and system execution returns to a "safe state" before proceeding. This paper focuses on run-time performance.

Most techniques for development of fault tolerant software fall into one of two broad classifications: redundancy and exceptions. Redundancy is often implemented as N-version programming, or NVP [1]. The Ada tasking model is particularly appropriate for embedded real-time systems with only a run time system and no intervening operating system. Clearly, distributed or parallel environments are more appropriate for NVP than sequential ones, although network connection can fail or be slowed down by an unexpected excess of traffic. Virtual environments such as VMware also can be useful.

Exception handlers are often known as recovery blocks, or RB [6]. Assertions and guards, which are source code statements within a block of code that a condition holds, also fall into this category. The assertion is verified to be correct before the block of code is executed. Assertions are most common in programs written in C or C++. Unfortunately, the testing of an assertion is not always an atomic operation, even on a single machine.

The proper use of exceptions requires three things: detection of the exception by some routine, sending the exception to another routine, and the handling of the exception. Many modern programming languages support some form of exception, but their run-time systems may be different, especially in the determination of where program control resumes after an exception occurs and is detected. Exception handlers may be searched for in local programming blocks, in encapsulating procedures or tasks, and so on, sometimes with control returning directly to the operating system.

Detection and handling of exceptions are not always atomic operations, causing system inconsistency after a fault. For distributed systems, coordinated atomic actions may be required.

2. Predictable overhead of FT

There is overhead in the design and implementation of voters, acceptance tests, and in the maintenance of safe states. Voters and acceptance tests can be developed: as a separate task implemented in Ada, as a normal (heavyweight) process in C or C++, as a (lightweight) thread on a single computer in Java, as a thread on a remote computer, or similar. A safe state can be a configuration file, a permanent entry in a log file, or a dynamic entity. Each design decision will impact the run-time overhead of the system.

It is easy to evaluate the overhead of NVP implementations in most programming languages if the voter is well-designed. Examples of measuring such overhead in C and Ada are given in [3] and [4].

A voter implemented as an Ada task, with each version also implemented as an Ada task, can use the Ada rendezvous mechanism to handle tasks that do not produce results within expected times using, say, the selected wait mechanism. However, there may be limitations caused by implementation details. If the software runs directly under the control of the Ada run-time system, all resources may be made available.

There may be additional constraints if an Ada system runs on top of another operating system such as Linux (including UNIX variants) or Windows. A

voter's performance may be dependent on a specific implementation. Run-time performance may be slower, potentially causing timing and performance problems. Similar arguments hold for acceptance tests.

A voter that runs as a (heavyweight) process on a single machine will typically have a closer association with the native operating system, but the communications facilities may be more complex, especially for processes that use UNIX ipc.

A voter that runs as a (lightweight) thread on a single machine will typically have a closer association with the native operating system, and perhaps the run-time support system of a language such as Java, but the communications facilities may be more fragile at times than those of heavyweight processes.

3. The less predictable overhead of FT

Information on fault-tolerant performance may be needed during design. A performance estimation approach based on benchmarks that consider how large a load a system can handle, rather than how fast it can solve standard problems, can help estimate the overhead of fault tolerance techniques at design time.

Chen and Patterson introduced the concept of such self-scaling benchmarks and applied them to the evaluation of UNIX file systems [2]. Their benchmark code is highly portable to UNIX-based systems and can help immensely in eliminating inappropriate fault tolerance techniques during the design phase.

Their approach is well suited to fault tolerance. Determining how large a problem a system can handle is a critical issue in evaluating the system's performance under load such as a denial of service attack. We have extended their results to self-scaling benchmarks to evaluate UNIX ipc performance [5]; network extensions are easy. Our benchmark code can be changed to include any software components or objects that might be used as redundant versions, voter processes, exception handler, or acceptance tests.

4. The special case of COTS

Systems that use commercial, off-the-shelf (COTS) products often have both high- and low-level interactions. The development of wrappers often takes care of the higher level interactions, assuming that there is sufficient similarity in programming languages and operating system support and documentation is adequate.

Lower-level interactions are far more complex, because they are rarely described formally as part of a COTS product's external interface. Low-level interactions may include race conditions or access to a shared resource controlled by a non-atomic function.

COTS products can also acquire too many system resources, potentially causing faults. For example, there are four types of limits relevant to the number of open files within a UNIX process:

- A limit on the number of open files may be specified by a programming language such as ANSI C.
- A limit on the number of open files may be specified by a compiler.
- A limit on the number of open files may be specified by a configurable constant, often known as a "soft limit," specified in the file limits.h.
- An absolute limit, often known as the "hard limit."

Often, COTS products can be inserted into schemes for self-scaling benchmarks.

Acknowledgement

This research was partially supported by the National Science Foundation under grant number 0324818.

References

[1] Avizienis, A., "The N-Version Approach to Fault-Tolerant Software," *IEEE Trans. Softw. Engr, Vo*l. SE-11, No. 12, Dec. 1985, pp. 1491-1501

[2] Chen, P. M., and. D. A. Patterson, "A New Approach to I/O Performance Evaluation-Self-Scaling Benchmarks: Predicted I/O Performance," *Performance Evaluation Review*, Vol. 21, No. 1, 1993, pp. 1-12.

[3] Coleman, D. M., and R. J. Leach, "Performance Issues in C Language Fault-Tolerant Software," *Computer Languages*, Vol. 14, No. 1, 1989, pp.1-9.

[4] Leach, R. J., and D. M. Coleman, "N-Version Programming using the Ada Tasking Model," *Proc. Ninth Annual National Conference on Ada Technology*, March 4-7, 1991, Washington, DC, 135-141.

[5] Leach, R. J., and Q. Zhang, "Self-Scaling Benchmarks of Interprocess Communications Subsystems," in preparation.

[6] Randell, B., "System Structure for Software Fault Tolerance," *IEEE Trans. Softw. Engr*, Vol. SE-11, No. 2, June 1975, pp. 220-232.

[7] VMware, VMware Infrastructure Data Center Management and Optimization Suite. Article on-line. Available from http://info.vmware.com. 2007.

Towards Concurrent SLA-based Management in a Composite Service Data Centre

Jim Smith and Paul Watson
School of Computing Science, Newcastle University, UK
Jim.Smith@ncl.ac.uk,Paul.Watson@ncl.ac.uk

Abstract

The ongoing aim behind the work described here is to investigate support for a data centre type environment where an application can be implemented as a composition of components, or workflow. For instance, an application might reuse established components. As is typical in data centre operation, hosting of an application is governed by a Service Level Agreement (SLA) between provider and application owner. Such an SLA describes levels of service and corresponding charges (paid to the provider) and refunds (paid by the provider). The infrastructure can of course vary mapping of hosted applications to machine resources in seeking to meet SLAs efficiently. However, it is also possible to dynamically vary the mapping of composite application to components; for instance to switch between alternate implementations of some particular component. The suggestion is that the availability of alternative components, or even compositions, will arise naturally in a shared repository, and that the presence of the extra degree of control can make the application hosting more resilient. While previous work by the authors has demonstrated basic mechanisms towards a composite service data centre, the current work begins to combine such mechanisms towards the control of both resource and component mapping in the context of concurrent workloads. The paper describes experimental work using a prototype implementation of an adaptive workflow engine.

1 Introduction

In common with other component based systems, service oriented computing aims to facilitate reuse of established components, thereby both saving on development effort and pushing application design to a higher level of abstraction where errors are less likely. Thus, the services offered by an organization can be represented as machine readable interfaces (WSDL). These are stored in registries with associated metadata describing functional and non-functional properties, where they can be found so as to support use of the underlying service by client applications. Thus it is common for complex applications to be implemented as composite services (aka workflows) linking services at widely distributed locations. In a distributed setting, it is to be expected that one execution of such a complex application may differ from a previous one, e.g. through using different services or service instances; even that a composition may be dynamically altered during execution, e.g. in the event of a failure. To this end, the client may specify a set of requirements, which may be encapsulated in a service level agreement (SLA). For instance, [15] demonstrates composition which is dynamic in respect of non-functional attributes such as duration, price and availability. However, it is also possible to consider composition in respect of functional requirements too, e.g. [10]. Such approaches tend to be concerned with adapting an individual workflow.

In recent years there has been a notable tendency for organizations to out-source some or all of their IT applications, and several companies have developed large data centres to host the client applications. Thus, work such as [3] demonstrates that multiple applications can be hosted dynamically and securely, but a need arises for the improvement of management algorithms if utility computing is to achieve efficiency [2]. Recent work, e.g. [13], has shown good scalability in terms of mapping monolithic applications. Also, while the inherently multi-dimensional nature of SLAs is well appreciated [9], it appears to be typical for experiments in data centre control to emphasize user response time. The notion of a shared repository in which applications may be developed, potentially exploiting reuse, and hosted, can be seen appearing in offerings such as Amazon Web Services [1], and CARMEN [5].

Variation of of component replication is used in support of meeting SLAs for composite service invocations in [11]. In general there can be bounds on useful replication, e.g. due to database access [4]. In the context of multi-tier web server applications, [14] addresses the problem of taking account of bounds on parallelism in those tiers in or-

der to use data centre resources efficiently. Such concerns are significant too in the composite service data centre of this work, but this paper is more closely related to a scenario described in [6] where web server applications are dynamically switched between levels of quality (full graphics and text only) in order to meet response time requirements. Specifically, the experiment described here demonstrates that dynamic swapping of component implementations can be used to meet multi-dimensional SLA requirements.

2 Adaptive and Concurrent Management of Composite Services

A prototype system has been implemented by wrapping an established 'basic' workflow engine [7] in "black-box" fashion, as shown in Figure 1. At deployment, a workflow is

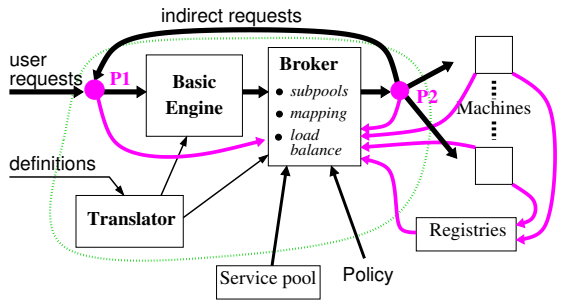


Figure 1. Adaptive Management of Composite Services

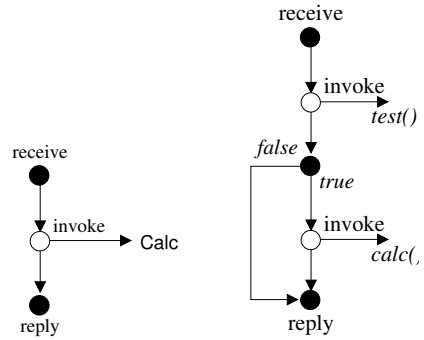
modified by a **Translator** to add extra identification parameters to each component invocation. Each component invocation from the deployed composite service is intercepted by a **Broker**, which uses these identification parameters and statistics forwarded by probes, **P1** and **P2**, to determine, in accordance with a chosen policy, how to process that request. The **Broker** dynamically partitions the pool of available machines into a number of subpools, according to the number of workloads, and moves machines between subpools in accordance with target sizes which are determined on the basis of the probe measurements. The **Broker** also constructs an invocation map for each deployed workflow, determining between alternate component services again on the basis of the measurements reported by the probes. Further details of the prototype may be found in [12]. In the current implementation, a policy is implemented directly as Java code within a controller class within the **Broker**.

In this experiment, the policy seeks (approximately) to maximize revenue for the provider. The first mechanism employed is a slight adaptation of the *measured loads* heuristic of [8] in selection of subpool sizes. This heuristic follows from the well known queueing result of Little which equates offered load in a queueing system to the product of arrival rate and response time. Both workflow arrival rate and workflow response time are measured for each workload in probe **P1**. For a given interval, the heuristic partitions the pool of machines in proportion with the product of the offered loads and workload revenue coefficients; the former measured during the previous interval. The slight adaptation to the previously described heuristic is in the definition of the cost coefficient. In the experiment described here, the coefficient is computed as the square of the average of possible *refund* values divided by the average of possible *charge* values, rather than simply *refund* divided by the *charge*. The idea behind squaring the *refund* value is to ensure that few resources are allocated to a less profitable workload. The use of averaging reflects the presence of alternate levels of charge or refund, depending for instance on result quality. A second mechanism is employed to control the response time where necessary in the scenario where a lower quality answer on-time is preferable to a higher quality answer late. Specifically this mechanism switches between maximizing a named attribute, e.g. result quality, and minimizing response time for each component service (measured in probe **P2**) with alternative implementations within a composite service, depending on whether the measured average response for the composite service is (respectively) less than or greater than a defined bound. Effectively, one non-functional attribute is traded-off with another. The longer term idea is to decide between admission control, i.e. rejection of requests and quality downgrade dynamically, and in the case of the latter to decide dynamically which attributes to downgrade. The experiment described here demonstrates a particular example scenario.

3 Experiment

In the experiment described here, as in [12], the two example workflows shown in 2 are used. Both *OneCall* and *TwoCallAnd* invoke a service *Calc*, but in the latter case, *Calc* is only invoked if the prior call to *Test* returns *true*. Both *Test* and *Calc* are implemented as configurable CPU-heavy loads. Each has a single operation, *test* and *calc* respectively, which is invoked in these experiments. As in the earlier work, two implementations of the more expensive service are defined with differing properties, as shown in Table 1. The execution cost of the *test* operation in this work is equal to that of the cheaper implementation of *calc*. The actual execution times of single invocations of the two *calc* implementations in this experiment are 4.7 and 47 seconds.

There are many possible definitions for an SLA which could reflect the various possible outcomes of a request. For



(a) *OneCall* (b) *TwoCallAnd*

Figure 2. Example workflows.

	monetary cost	result precision	run-time
Calc000	4.5	10	× 1.0
Calc001	0.5	1	× 0.1

Table 1. Alternative service implementations.

this initial experiment, the SLA is assumed to have the form shown in 3 are used. In the context of the example work-

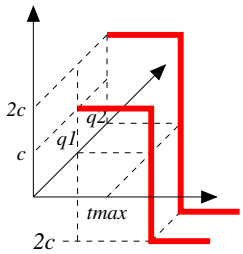


Figure 3. Example SLA agreement.

flows there are just two possible values for the overall result quality, shown as *q1* and *q2*. In the example workflows, the result quality is purely dependent on the choice between the two implementations of *Calc*. For the sake of simplicity, there is a single response time bound *tmax*, below which one of two charges, *c* or *2c* is payable to the provider, but above which a single refund *2c* is payable by the provider.

In the experiment, there are three concurrent workloads; workload *A* comprises requests against *OneCall* and the other two, *B* and *C*, make requests against *TwoCallAnd*. *TwoCallAnd* is configured such that *test* returns *true* in 50% of requests; in just those requests is *calc* invoked. The workflow requests in all three workloads follow Poisson sequences. In the case of workload *A* which submits 500 requests against *OneCall*, the Poisson rate is increased from 0.067 to 0.155 at time 2000 seconds. In the case of workload *B* which submits 1000 requests against *TwoCallAnd* (recall that 500 of these invoke the expensive operation), the Poisson rate is decreased at time 2000 seconds from 0.38 to 0.1125. In the case of workload *C*, the 250 requests against *TwoCallAnd* are submitted at a constant Poisson rate of 0.0565.

The SLA for each of the workloads has the form shown in Figure 3. However, while *A* and *B* have *c*= 0.5, *C* has *c* = 0.05. Intuitively then, the system should allocate most resources to *A* and *B* and potentially switch between implementations of *Calc* in order to control the average response time, particularly in the case of workload *C* which should be squeezed in terms of resources.

4 results

The experiment is conducted in a Linux environment. The pool comprises 20 2.8GHz machines in a cluster. The workload generators and adaptive workflow engine are run on a separate 3GHHz machine. Figure 4 shows how the actual pool sizes vary during the workload. As expected, most

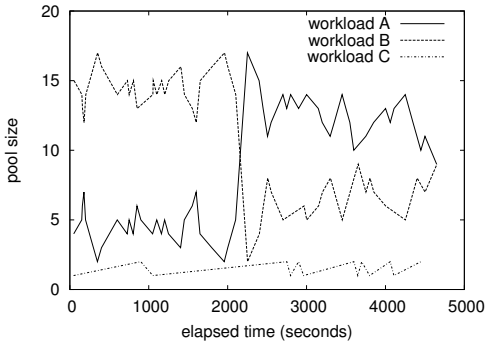


Figure 4. Recorded pool sizes.

of the machines are allocated to the two higher value workloads. The significant transfer of machines between these two workloads corresponds to the change, at time 2000, in request submission rates described earlier.

Figure 5 shows the measured durations of individual requests in each of the three workloads. In the case of workload A, there is a period of settling at startup, during which 6 requests are mapped to the cheaper alternative of *Calc*. Following this, the response times for requests in workload A are mostly constant, and close to the expected value. Those in workload B, are mostly split evenly between about 5 and about 50 seconds, reflecting the ratio of requests which make an invocation of *calc*. In practice, all of the *calc* invocations in workload B, and all apart from the 6 requests mentioned above in workload A are mapped

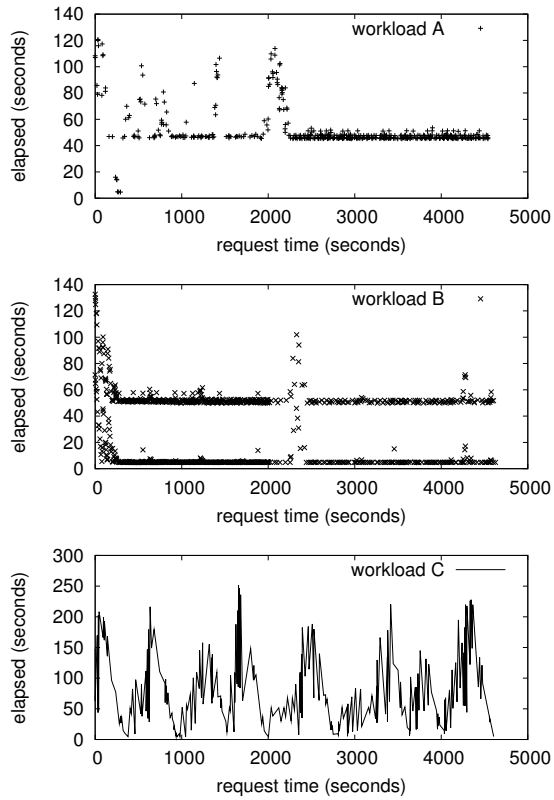


Figure 5. Measured request times.

to the more expensive implementation of *Calc*, reflecting the higher value of these workloads in revenue terms. By contrast, the response times in workload C oscillate about the chosen switching value, which was set to 90 seconds in this experiment. In practice, out of the 144 invocations of *calc* in workload C, 87 were mapped to *Calc000* and 57 to *Calc001*. For the lower value workload, which is squeezed for resources, the service mapping is dynamically mapped between cheaper and more expensive implementations, in order to trade quality so as to keep the average response time close to that specified in the SLA.

5 Conclusion

An experiment in the management of concurrent workloads is presented, where the workloads invoke composite services which are mapped dynamically onto a shared pool of machine resources and whose component invocations are mapped dynamically between alternate implementations. It is anticipated that dynamic composition of this nature might become a feature of shared repositories which are starting to appear. The results presented here are limited in several ways, but demonstrate that it can be possible to employ dynamic service invocation mapping in support of concurrent SLA based management of composite services. Specifically, a lower value workload can be implemented at degraded quality. A somewhat simplistic approach is employed here which maps all invocations in a workflow for response time or quality. In general, this offers an alternative to rejecting either the workload or particular requests, and an obvious direction for ongoing work is to evaluate such alternatives. As hinted earlier of course, trading between non-functional properties can also be possible in the context of monolithic applications.

References

[1] Amazon. Amazon web services. http://www.amazon.com/gp/browse.html?node=3435361, 2008.

[2] A. Andrzejak, M. Arlitt, and J. Rolia. Bounding the resource savings of utility computing models. Technical Report HPL-2002-339, HP Labs, Dec. 2002.

[3] K. Appleby, S. Fakhouri, L. Fong, G. Goldszmidt, M. Kalantar, S. Krishnakumar, D. P. Pazel, J. Pershing, and B. Rochwerger. Oceano-sla based management of a computing utility. In *International Symposium on Integrated Network Management*, pages 855–868, 2001.

[4] A. Bartoli, R. Jimnez-Peris, B. Kemme, C. Pautasso, S. Patarin, S. Wheater, and S. Woodman. The adapt framework for adaptable and composable web services. *Distributed Systems On Line*, 2005.

[5] CARMEN. Code analysis, repository and modelling for e-neuroscience. http://www.carmen.org.uk/, 2008.

[6] S. Cheng and D. G. B. Schmerl. Architecture-based self-adaptation in the presence of multiple objectives. In *SEAMS*, pages 2–8. ACM, 2006.

[7] A. Endpoints. Activebpel engine. http://www.activebpel.org/, 2007.

[8] M. Mazzucco, I. Mitrani, and J. Palmer. Web service hosting and revenue maximization. In *ECOWS*. IEEE, Nov. 2007.

[9] C. Molina-Jimenez, J. Pruyne, and A. van Moorsel. The role of agreements in it management software. In *Architecting Dependable Systems III*, pages 36–58. Springer, 2005.

[10] S. Ponnekanti and A. Fox. Sword: A developer toolkit for web service composition. In *WWW*. IEEE, 2002.

[11] K. Ranganathan and A. Dan. Proactive management of service instance pools for meeting service level agreements. In *ICSOC*, pages 296–309, 2005.

[12] J. Smith and P. Watson. Experiments towards adaptation of concurrent workflows. In *ECOWS*. IEEE, Nov. 2007.

[13] C. Tang, M. Steinder, M. Spreitzer, and G. Pacifici. A scalable application placement controller for enterprise data centers. In *WWW*, pages 331–340. ACM, 2007.

[14] B. Urgaonkar, P. Shenoy, A. Chandra, and P. Goyal. Dynamic provisioning of multi-tier internet applications. In *ICAC*, 2005.

[15] L. Zeng, B. Benatallah, A. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. Qos-aware middleware for web service composition. *Transactions on Software Engineering*, 30(5):311–327, may 2004.

Session 2B: Fast Abstracts II

Security and Ontologies

EDCC-7

Setup Time Violation Attack on DES and Triple-DES

Ying ZHUANG and Nidhal SELMANE and Sylvain GUILLEY and Jean-Luc DANGER

Abstract

Eli Biham and Adi Shamir introduced a differential fault attack (DFA) on DES in 1997 [2]. This attack assumes that an attacker is able to induce a single bit-flip in a datapath register of DES. In this article, we present a concrete setup in which Biham and Shamir's fault model happens to be correct. This definitely proves that the original DFA on DES is practical. The same attack can be applied on implementations of 3DES algorithm.

In the proposed attack scenario, the faults are the consequence of a setup time violation on the combinatorial path of a hardware accelerator for the DES algorithm. Such errors are injected by supplying the DES coprocessor with a voltage supply inferior to its nominal value. We developed an algorithm to locate the spatio-temporal positions of the up to eight faults. The result is that single faults can be obtained over a wide range of voltage.

1. Introduction

Hardware implementation of cryptographic algorithm on smartcards are extremely reliable, however it is possible to intentionally induce faults into computations in order to retrieve secret information. There are several methods to induce faults on smartcards such as changing the power supply voltage, the frequency of the external clock or applying radiation. Until now, very few DFAs have been applied in real experiments. For instance, the paper [1] describes an experimental setup, and compares various methods of fault injection. However, the attack is only sketched superficially, without quantitative results.

In this paper, we first introduce the principle of differential fault attack on DES [5]. We also describe our fault hypothesis and the principle of setup time violation attack.

Secondly, we introduce our experiment, that includes an on-line and an off-line stage. We first describe how we acquire faulty ciphertexts, and then we discuss our methodology of fetching multiple errors (up to 8 on DES). In the last part of the paper, we show the coverage of different fault model and analyze the result of the experiment.

2. DFA on DES

DES is a 16-round secret key encryption algorithm based on a Feistel structure. The model of Biham-Shamir [2] lets the attacker induce random bit-flip errors during the DES encryption. In this attack, the adversary obtains several couples of ciphertext derived from the same plaintext and key, where one of the ciphertext is correct and the other is the result of a computation corrupted by a single bit error during the computation. Consequently, a single bit-flip fault occurring in the left register "L" will cause a single bit-flip in the right register "R" at the next round. Similarly, a single bit-flip fault occurring at the output of any non-linear substitution box (nicknamed sbox) will also translate into a single bit-flip in register R. Thus, it is tantamount to attack the registers, or the substitution boxes, or actually any part of the DES algorithm.

Faults are exploitable if they occurs on round 16, 15 or 14. By using faulty and correct ciphertext the attacker can retriever the last subkey. Then we can proceed according to two strategies: The first one consists in using the fact that this subkey contains 48 out of the 56 key bits in order to guess the missing 8 bits in all the possible $2^8 = 256$ remaining bits. The second one consists in using the knowledge of the last subkey to peel off the last round (and remove faults that we already identified) and to analyze the preceding rounds with the same data.

3. Errors Typology

The errors considered in this paper are those caused by an early latching of a combinatorial function. For the sake of simplicity, we assume that the errors appear in the datapath LR.

In the datapath LR, we call $E(R, S, M)$ an error located at round $R \in [1, 16]$, substitution box $S \in [1, 8]$ and causing a bit-flip on any or all of the sbox 4-bit outputs, resulting in its masking with $M \in \{0, 1\}^4 \backslash \{0\}^4$. The reason why in E several (from one to four) bits are affected is that we do not exclude the possibility to faults occur with the sboxes. Notice that the last gates of the datapath are the `XOR` gates situated downstream the sboxes. However, these

gates propagate the errors that might happen in the sboxes; thus, by considering errors in the sbox, both cases (error in sboxes and in the final XOR) are concomitantly captured. Incidentally, it will be impossible for an attacker to distinguish between the two cases when it is a single bit-flip (*i.e.* the masks M are restricted to the unitary Hamming weight subset of $\{0,1\}^4$.) This distinction is anyway irrelevant for attacks, because only the syndrome (and not what actually triggered it) is exploitable.

The DES blocks that are susceptible to setup time violations errors are highlighted in Fig. 1. The areas where errors are likely are highlighted.

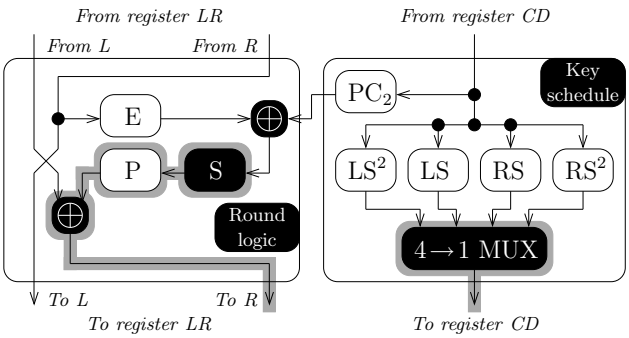


Figure 1. DES round and key schedule combinatorial logic.

Notice that, if the DFA requires an access to the encryption hardware to work, the attack in itself is totally independent from the hardware architecture. The datapath depicted in Fig. 1 is thus just an illustration.

However, in case of two different $E(R,S,M)$ errors, the attack can be accelerated.

Handling complex simultaneous errors is computationally intractable. In Tab. 1, the complexities of multiple errors are given.

4. Experimental Attack

The DFA proceeds in two phases:

1. an *on-line* faults collections from the targeted device, and

2. an *off-line* analysis of the faults collected with CFP algorithm.

These two phases are dealt with in the two following subsections.

4.1. Acquisition of the Faulted Ciphertexts

The experimental results presented in the sequel are specific to the circuit SecMat V1 [4], embedding an iterative DES co-processor whose architecture is described in [6]. However, related targets are expected to yield similar observations, since the error model is proved to take its root from two physical phenomenon (the dependence of the propagation time in the data and the metastability of the DFFs). Our experimental setup consists in a dual power supply:

- U is the general core voltage of the SoC.
- V is the dedicated voltage of the DES coprocessor.

We use the Agilent E3631A power supply to decrease the nominal voltage (1.2 V) by ½ mV steps for V as shown in figure 2. The apparition of errors in the device is linked to

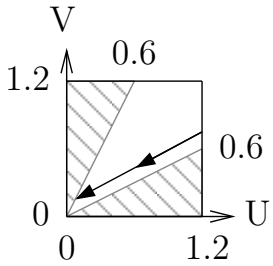


Figure 2. Power supply used to bring DES into a faulty behavior.

the clock period T, as shown in the Fig 3. Each point of the figure is obtained by the averaging over 2 500 encryptions, for a constant message and a constant key. It appears that within about 10 mV, the device gradually moves from an error-free to a fully erroneous behavior. This means that an attacker can choose reproducibly the probability of the faults by tuning the voltage. When the frequency is high, the faults start appearing at a higher voltage than at low frequency. The reason is that when the supply voltage V falls, the propagation time $T_{propagation}$ rises.

We notice that some errors are redundant so we can define a new metric to characterize the fault unicity. It is called "unique errors ratio $\doteq \frac{\text{unique faults}}{\text{total faults}}$", and illustrated experimentally in the Fig 4. We notice that at low frequency (32 MHz), the variability of the faults is low. This can be explained by the fact that at low voltages, the critial paths are more scattered. Therefore, the same paths is likely to be violated repeatedly, hence a little unique errors ratio.

4.2. Principle of the CFP Algorithm

In order to identify the multiplicity of faults, we developed an algorithm called CFP (short for "Ciphertext

Table 1. Number of possibilities of multiple-faults in the DES algorithm.

Error type	Number of possible occurrences		
E^1	$1\,920$	=	$16 \times 8 \times \left(2^4 - 1\right)$
E^2	$3\,684\,480$	=	$\left(\#E^1 - 0\right) \times \left(\#E^1 - 1\right)$
E^3	$7\,066\,832\,640$	=	$\left(\#E^1 - 0\right) \times \left(\#E^1 - 1\right) \times \left(\#E^1 - 2\right)$

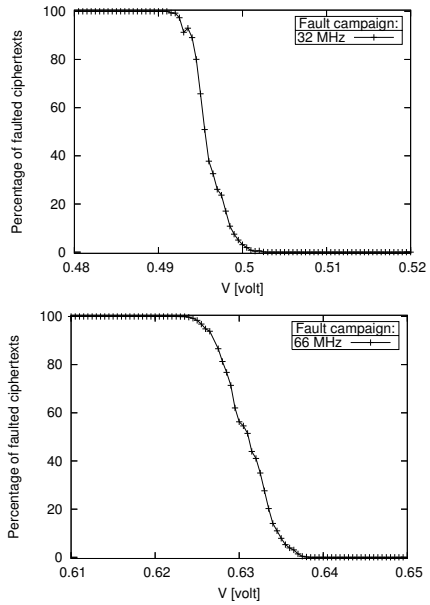


Figure 3. Percentage of faulted ciphertext running at $T = 32$ and $T = 66$ MHz.

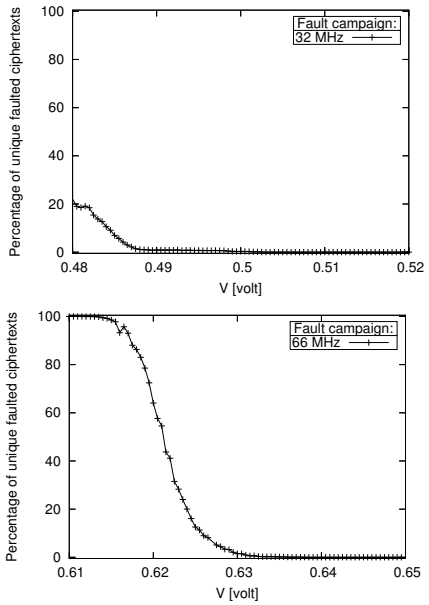


Figure 4. Percentage of unique faulted ciphertext running at $T = 32$ and $T = 66$ MHz.

Fault-Positions"). It is used to find out fault positions of the fault-ciphertext generated by reducing the supply voltage and saved in the acquisition files. This algorithm contains 2 functions. One is called $Dict()$ that is used to create dictionaries that present the all of the fault-position possibilities by the couples of fault-ciphertext and fault-positions. Another is called $Compare()$ that is used to compare the dictionary and the acquisition files to locate the fault-ciphertexts.

The algorithm CFP is executed by iteration, that is CFP^n will take the fault positions list called FPL^{n-1} generated by CFP^{n-1}. The most fundamental dictionary is that corresponding to one single fault. By scanning all of the possibilities of position, 16 rounds, 8 sboxes, with all of the mask values $(2^4 - 1)$, we could have a dictionary $Dict^1$ presenting the "fault cipher-text and the fault position". Comparing $Dict^1$ with the acquisition files, we will find the all of the fault-ciphertexts generated by a single fault. We save this result in a log file Res_file^1, and these single faults will be saved in a fault-positions list called FPL^1.

To create the dictionary $Dict^2$ for 2 faults, we will choose one fault-position (we call it $fixed\_error$) in FPL^1, the other one, we will take it by scanning all the possibilities (all rounds, all sboxes and all the values of mask). After comparing the acquisition files with the $Dict^2$, we could find out the fault-ciphertexts generated by 2 faults in the acquisition files. If there are new fault positions that are not in FPL^1, we will add them in it, that is the fault-positions list is update to FPL^2. For $Dict^3$, the 2 $fixed_error$s will be chosen in FPL^2, another one will be token by scanning all the possibilities. The following is in the similar way. At last, we will have a final fault-positions list and the result files classed by number of the fault.

To create dictionary, the function $Dict()$ choose $n - 1$ $fixed_error$s in fault-positions list by recurrence to make up n errors and then construct a dictionary $Dict^n$.

We have written another function to calculate the coverage of the non-repetitive fault-ciphertexts found and multi

fault-ciphertexts at the same supply voltage by counting the number of the fault-ciphertexts found at this voltage. Then we obtain a plot like Fig 5.

5. Errors Characterization

Amongst the faults actually observed, those that fall into a proposed error model are said to be covered. The recognized errors coverage is plotted in Fig. 5. The left graph represents the coverage of non-repetitive faults and the right one is the coverage of repetitive faults. A filtering of the stuck-at-zero errors allows to improve the coverage.

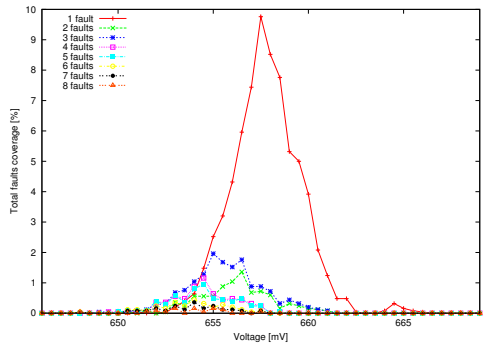


Figure 5. Total faults coverage at 66 MHz.

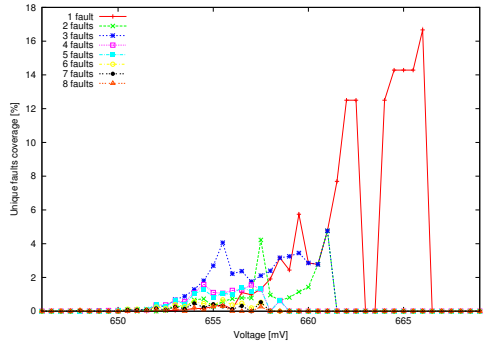


Figure 6. Unique faults coverage at 66 MHz.

According to the figure 6 we are able to collect a lot of unique errors, which is important to realize Biham's DFA, because it needs between 50 and 200 unique faults to succeed the attack.

In both figures 5 and 6, it happens that the single fault model is the most likely: its expectancy, equal to area of its density of probability, is greater than any other fault model. The single fault manifests first, and multiple faults follow suit when V is low. This validates the fact that single bit-flips in the R register is a realistic fault attack model on Feistel encryption schemes. The attack of Biham and Shamir [2] is thus practical on devices where the supply can be freely tampered with.

In the figure 6 showing repetitive faults, the coverage can be reduced down to 0. The reason is that the faults multiplicity is increasing quickly when the supply voltage drops.

6. Conclusions

This article reports an experimental setup to induce faults in a DES co-processor. The proposed setup is very low-cost, and still proves to be very efficient. We demonstrate that E. Biham and A. Shamir's differential fault attack can indeed be realized in practice. In addition, we provide with the comprehensive analysis of the faults' physical origin. We notably argue that their randomness is due to the metastability of the DFFs that sample the critical path.

Some asynchronous architectures suitable for secured implementations have been proposed [3]. These architectures are probably more immune to the setup time violation attack proposed in this article than their synchronous counterparts, because there is no implicit assumption that a computation is over (like it is the case for synchronous logic). Of course, this kind of attack could be prevented by a frequency-detector or by the internal generation by a PLL of the clock signal. So the targets of DFA attack are the synchronous systems in which the frequency-detector might be destroyed or the unprotected systems such as those carelessly implemented in FPGAs.

References

[1] Bar-El, H. and Choukri, H. and Naccache, D. and Tunstall, M. and Whelan, C. The sorcerer's apprentice guide to fault attacks. *Proceedings of the IEEE*, 96(2):270–382, February 2006.

[2] E. Biham and A. Shamir. Differential Fault Analysis of Secret Key Cryptosystems. *CRYPTO 1997*, 1294:513–525, 1997.

[3] G. Bouesse, M. Renaudin, B. Robisson, E. Beigné, P.-Y. Liardet, S. Prevosto, and J. Sonzogni. DPA on Quasi Delay Insensitive Asynchronous Circuits: Concrete Results. In *XIX Conference on Design of Circuits and Integrated Systems, Proceedings of DCIS'04)*, 24–26 Nov 2004. Bordeaux, France (PDF).

[4] "Circuits Multi-Projets" (alias CMP, < cmp@imag.fr >) Annual Report 2005. (Online PDF version).

[5] NIST/ITL/CSD. Data Encryption Standard (DES). FIPS PUB 46-3, Oct 1999.

[6] Sylvain Guilley and Philippe Hoogvorst and Renaud Pacalet. A Fast Pipelined Multi-Mode DES Architecture Operating in IP Representation. *Integration, The VLSI Journal*, 40(4):479–489, July 2007. DOI: 10.1016/j.vlsi.2006.06.004.

Malicious fault characterization exploiting honeypot data

Corrado Leita, Olivier Thonnard
Institut Eurecom
France
leita@eurecom.fr,
olivier.thonnard@rma.ac.be

Eric Alata
LAAS
France
ealata@laas.fr

Marco Serafini
TU Darmstadt
Germany
marco@deeds.informatik.tu-darmstadt.de

Vladimir Stankovic
City University London
United Kingdom
ek274@soi.city.ac.uk

Jouni Viinikka
France Telecom R&D
France
jouni.viinikka@orange-ftgroup.com

Urko Zurutuza
Mondragon University
Spain
uzurutuza@eps.mondragon.edu

1 Introduction

ReSIST is a NoE that addresses the strategic objective "Towards a global dependability and security framework" of the European Union Work Programme, and responds to the stated "need for resilience, self-healing, dynamic content and volatile environments". In the context of the Resist Network of Excellence, funds were allocated to one-year sub-projects addressing some of the "dependability and security gaps" identified by the network activity. We introduce here the outcomes and deliverables expected by one of these sub-projects, named "Honeypots", started in January 2008 and with expected completion at the end of 2008.

In order to assess the resilience of a system it is important to take into consideration malicious faults. In the specific case of a computer system, one of the most important classes of malicious faults is linked to Internet attacks. The gap GA2 in [13] shows how the lack of unbiased, representative and useful data poses a serious limitation to any attempt of quantifying and characterizing the security threats to which a computer system may be exposed. Despite the considerable number of ongoing projects [17, 3, 4, 1, 6] aiming at collecting data about Internet malicious activities, the lack of "good" data with respect to this objective is an important gap of the current state of the art. Also, there is a lack of rigorous methodologies and models that can be used to extract relevant information and trends from the collected data.

In this project we want to address an interesting sub-problem of this gap: classify and characterize real-world Internet attacks linked to the spread of self-propagating malware. While the scenario of Internet attacks in the past was characterized by a few but extremely visible phenomena linked to the spread of worms, nowadays' scenario is far more complex. The objectives of the hackers have changed, and what started as a competition to gain visibility by infecting large populations of machines in very short time has now become a more organized covert activity aiming at economical profit. Internet is now dominated by a large number of different classes of malware aiming at silently taking control of computer systems in order to steal banking accounts, generate spam, steal credit card numbers, and many other profit-oriented operations. Even if the presence of these different classes is of public knowledge, we lack quantitative information on their spread, on their impact, on their evolution. This project aims at investigating the feasibility of using real data provided by honeypots to obtain such information.

2 Project activity

We plan to collect data taking advantage of a distributed honeypot deployment, developed at Institut Eurecom, called SGNET [8, 9]. SGNET aims at deploying small sensors in different locations of the IP space. These sensors emulate network services and, taking advantage of the ScriptGen technology [10, 11], achieve a very high level of verbosity at

a very low cost. With respect to existing honeypot deployments, SGNET allows to achieve an extremely rich amount of information about the network activities hitting the honeypot, with special focus on code injection attacks. SGNET is in fact able to emulate the whole attack trace, understanding the presence of a successful code injection, and emulating its behavior ultimately downloading samples of the malware. This process allows to retrieve extremely valuable information about the intention of the attacker and its nature. This information is stored in a relational database for further analysis and constitutes the foundation over which we are planning to work within this project.

The "Honeypots" project aims at exploiting the information collected by the SGNET deployment. The project activity will follow two main lines of operation. Firstly, we will enrich the collected data through the comparison with other information sources. Secondly, we will dig into the collected data and try to abstract meaningful information taking advantage of existing algorithms developed by the participants in different contexts.

2.1 Correlation with other information sources

Some of the participants to the project manage different honeypot deployments taking advantage of different technologies. These technologies range from Honeynet Alliance deployments [1] to other high interaction techniques [2]. The reasons underneath the comparison of the information collected by different techniques are twofold. On the one hand, the diversity in collection techniques may allow to detect "blindness" of one approach to security events observed by the other approach. On the other hand, combining diverse information retrieved using different approaches potentially allows to enrich and consolidate the global view on the observed security events.

City University has deployed several high-interaction honeypots, which contain a mixture of Linux and Windows hosts and following the directions of the Honeynet Alliance [1]. These deployments emulate the traffic observed in corporate or SME (Small to Medium Business) environments. The network traffic collected in these honeynets will be compared with the SGNET data, e.g. exploratory analysis of the times between attacks on different hosts, operating systems, networks or geographical location will be detailed.

Mondragon University will proceed on the same line of City University, but from a different perspective. Mondragon University will provide a very rich dataset collected by around 10 honeypots of the Basque Honeynet Alliance [12]. This will allow to a very interesting comparison with the information provided by the SGNET deployment. The comparison will focus on different parameters than those analyzed by City University. For instance, both honeypot deployments being able to download malware using different strategies, we will compare the trends in malware download offered by the two technologies.

LAAS University will exploit the previous work on high interaction honeypots [2] to observe the behavior of attackers once they manage to get access to the core of the system. We plan to use this information to be able to provide answers to fundamental questions such as "what attackers do in the core of the system?" And "what are their objectives?". The goal is to better understand the motivations and methods of attackers. One way to achieve this goal is to study the sequence of commands performed by the attackers, using clustering techniques and symbolic time series.

2.2 Data enrichment through analysis

The information retrieved by the honeypot deployments is vast. In order to build characterizations of the observed security events we need a set of tools and techniques to mine the collected data and extract more meaningful aggregate information. We plan to take advantage of existing algorithms developed by some of the participants to the project to achieve this goal. The data mining techniques developed in the context of the Leurré.com honeypot deployment [5] can be of great help in organizing the collected data. Time series analysis developed in the context of IDS alert analysis [16] and statistical modelling [7] can help us in digging into the data filtering out uninteresting activities.

The data mining algorithms developed at Eurecom have been used so far to discover similarity patterns between the attacks over extended periods of time (e.g. several months or years). This long term analysis has already delivered interesting results about certain classes of malicious activities that seem to be correlated over long periods of time. In order to better address the highly dynamic and changing behavior new emerging threats, and also to understand how the threats evolve over time, we need to effectively identify the relevant periods of attack activities (i.e. the "attack events") on our sensors prior the execution of any correlative analysis. France Telecom R&D will provide its expertise in time series analysis [15, 14] to identify attack events within the honeypot time series. This can increase the quality and the significance of the results of the correlation algorithms developed by Eurecom. The final objective is to combine and automate both approaches (detection and correlation) so as to facilitate the processing of large quantities of data and to improve the discovery of meaningful information from honeypot data.

Eurecom will also provide an extension of its correlation framework in order to take advantage of the extensive information provided by SGNET. Such an extension will be used in the second half of the project to conduct an extensive analysis, so as to understand and characterize the evo-

lution of malware activities in both the time domain and in the "spatial" domain (i.e. along the Internet IP space).

Attackers can use intruded nodes to disrupt the operation of distributed systems, for example by creating inconsistencies in a distributed file system where data is replicated on multiple machines to guarantee integrity and availability. Since tolerance to intrusions (or in general Byzantine faults) is expensive, it is important to estimate how many faults are to be tolerated. TU Darmstadt will look at the Honeypot data to answer this question for systems that are subject to untargeted attacks.

3 Conclusions

Summarizing, the "Honeypots" project aims at exploring the feasibility of characterizing Internet attacks by digging into the information collected by different honeypot deployments. We hope to achieve this through the joint efforts of researchers with different competencies and resources. The ultimate goal of the project is providing a significant step towards the collection of unbiased, representative and useful data on real world attacks.

Acknowledgment

This work is supported by the ReSIST Network of Excellence (contract number 026764).

References

[1] Know your enemy: GenII honeynets. *Know Your Enemy Whitepapers*, May 2005.

[2] E. Alata, V. Nicomette, M. Kaâniche, M. Dacier, and M. Herrb. Lessons learned from the deployment of a high-interaction honeypot. In *EDCC'06, 6th European Dependable Computing Conference*, Oct 2006.

[3] M. Bailey, E. Cooke, F. Jahanian, J. Nazario, and D. Watson. The internet motion sensor: A distributed blackhole monitoring system. In *12th Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, February 2005.

[4] Caida Project. The UCSD Network Telescope, www.caida.org, 2007.

[5] M. Dacier, F. Pouget, and H. Debar. Leurre.com: On the advantages of deploying a large scale distributed honeypot platform. In *Proceedings of the E-Crime and Computer Conference 2005 (ECCE'05)*, Monaco, March 2005.

[6] DShield. Distributed Intrusion Detection System, www.dshield.org, 2007.

[7] M. Kaâniche, E. Alata, V. Nicomette, Y. Deswarte, and M. Dacier. Empirical analysis and statistical modeling of attack processes based on honeypots. *WEEDS 2006-workshop on empirical evaluation of dependability and security (in conjunction with the international conference on dependable systems and networks,(DSN2006)*, pages 119–124, 2006.

[8] C. Leita and M. Dacier. SGNET: a worldwide deployable framework to support the analysis of malware threat models. In *Proceedings of the 7th European Dependable Computing Conference (EDCC 2008)*, May 2008.

[9] C. Leita and M. Dacier. SGNET: Implementation Insights. In *IEEE/IFIP Network Operations and Management Symposium*, April 2008.

[10] C. Leita, M. Dacier, and F. Massicotte. Automatic handling of protocol dependencies and reaction to 0-day attacks with ScriptGen based honeypots. In *RAID 2006, 9th International Symposium on Recent Advances in Intrusion Detection, September 20-22, 2006, Hamburg, Germany - Also published as Lecture Notes in Computer Science Volume 4219/2006*, Sep 2006.

[11] C. Leita, K. Mermoud, and M. Dacier. Scriptgen: an automated script generation tool for honeyd. In *Proceedings of the 21st Annual Computer Security Applications Conference*, December 2005.

[12] Mondragon University. Euskalert, the basque honeynet alliance. Web page at `http://www.euskalert.net`, 2008.

[13] ReSIST NoE. From resilience-building to resilience-scaling technologies: Directions. *Deliverable D13*, 2007.

[14] J. Viinikka. *Intrusion Detection Alert Flow Processing Using Time Series Analysis Methods*. PhD thesis, University of Caen, Caen, France, Nov. 2006.

[15] J. Viinikka, H. Debar, L. Mé, A. Lehikoinen, and M. Tarvainen. Processing intrusion detection alert aggregates with time series modeling. *Information Fusion Journal*, 2008. Special Issue on Computer Security, to appear.

[16] J. Viinikka, H. Debar, L. Mé, and R. Séguier. Time series modeling for IDS alert management. *Proceedings of the 2006 ACM Symposium on Information, computer and communications security*, pages 102–113, 2006.

[17] D. Zamboni, J. Riordan, and Y. Duponchel. Building and deploying billy goat: a worm-detection system. In *18th Annual FIRST Conference*, June 2006.

AROVE-v: Assessing the resilience of open verifiable E-voting systems

Philippe Palanque, Marco Winckler, Regina Bernhaupt
IHCS group, IRIT, Université Paul Sabatier, Toulouse, France
{palanque, winckler, bernhaupt}@irit.fr

Eugenio Alberdi, Lorenzo Strigini
Centre for Software Reliability, City University, London, UK
{eugenio, strigini}@csr.city.ac.uk

Peter Ryan
School of Computing Science, University of Newcastle, Newcastle, UK
peter.ryan@ncl.ac.uk

Abstract

The work described in this paper addresses resilience issues in E-voting. An electronic election involves a large, complex, integrated, socio-technical system, encompassing large numbers of users and machines in different tasks, and having to satisfy multiple and complex requirements. Designing such a socio-technical system requires trade-offs amongst many interdependent factors such as dependability and security, trust, usability and privacy; assessing it requires consideration of complex and disparate evidence and arguments. This paper introduces a case study looking at a particular E-voting system ("Prêt-à-Voter") and describes one of the strands of work under way.

1. Introduction

Increasingly governments are moving towards electronic means to support the conduct of elections, whether this is in the vote capture or counting phases. There is also pressure to move to remote voting, internet, telephone etc. This has proved highly controversial, especially the prospect of remote voting.
Voting systems are socio-technical, critical systems, certainly when used for binding, political elections. They have to reconcile conflicting requirements of guarantees of accuracy and ballot privacy. Ideally we would like to achieve such assurance with minimal dependence on software, hardware, officials etc. This challenge has prompted the development of a number of so-called "end-to-end" voting schemes that strive to optimal transparency and auditability within the constraints of ballot secrecy.

When we talk about an "E-voting system", we refer to the large scale socio-technical system, involving vote-casting, ballot transmission and counting, auditing and monitoring systems. The E-voting system depends on a distributed network based service, on maintaining trust by the voters, which may be altered by perceptions of suspicious behaviour of the automated systems, on the availability of a distributed web-based service for monitoring ballot counting, and on distributed monitoring by many users. These systems include mechanisms that detect foul play with very high probability. However, the probability of successful completion of an election will be determined by people's proper reactions to mixes of false alarms and true alarms will determine. The collective behaviour of the system depends on local decisions (and errors) by many, diverse users, in polling station operations and in the monitoring phases. For instance, interfaces and procedures should ensure that random voter errors cannot produce systematic bias in election results. Responsibility for correct operation will be distributed between multiple organisations (e.g. government, parties, ISPs). Evidence to decide whether an E-voting system is dependable enough would come from disparate methods: from formal proof to statistical evidence of human behaviour, with difficulties from obvious constraints on direct measurement of voter behaviour.

Procedures for assessing these systems – for gaining, before use, assurance that they will work well,

and especially, resiliently – are still deficient, as highlighted by multiple stories, even very recent (e.g. problems in computer-assisted counting in recent Scottish election).

This abstract presents an ongoing case study, AROVE-v, conducted within the EU funded Network of Excellence ReSIST (on Resilience and Survivability for IST). The goal of AROVE-v is to identify necessary components of an "assurance case" or "dependability case" supporting the use of an E-voting system from the viewpoint of resilience, and recommend methods for establishing the necessary evidence. By "case" we mean a structured argument that integrates evidence about a system, assumptions and sub-arguments to support a claim about the system meeting specified dependability requirements. This approach comes from the area of safety assurance and there is growing recognition of its importance for all complex dependability assessments tasks [1].

The need for trustworthy dependability cases is particularly felt when the ability to predict future system behaviour is felt inadequate in view of the magnitude of the risks involved in operating the system. Hence the origin of the quest for "cases" in the area of safety. Large, integrated socio-technical systems often present a risk of very large losses, and difficulties of prediction linked to the imprecise boundaries of the systems, their potential for spontaneous evolution, the involvements of people in many roles, and often the difficulty of explicitly stating the dependability requirements themselves.

In the design and assessment of a system of these characteristics, e.g. e-gov systems [3], relevant factors have been identified as:

- **Usability**: poor usability would increase number of errors, reduce tasks performance, increase frustration, and result in the system being used less than it could be [4]. This might be even more critical if the system does not comply with accessibility guidelines, now widely advertised, and even a mandatory requirement in many countries that have enacted rules for Accessibility responsibility of content on the Web [5-8].
- **Security:** so that people (e.g. the voters) do not face problems of identity usurpation, or preventing malicious use of both system functionalities and stored data [9].
- **Trust,** which is built in the relation between service providers and users. One major element of the development and perception of trust relates to accountability which is usually based on an appropriate combination of legislation, industry self-regulation, and public oversight [10] but also requires that a design (of machinery and procedures) that supports accountability.
- **Privacy** protection: by preventing unauthorized access to private data either stored in a profiling system or in real time when citizens interact with the system [10]. It should cover concerns such as, network privacy, personal identification information privacy and preference or profile privacy.
- **Dependability:** so that their users can be ensured of both their reliability and their availability of service [12].

2. A Case Study: "Prêt-à-Voter"

AROVE-v focuses on a category of E-voting systems that are designed to deliver better privacy and tamper-resistance than paper-based elections without requiring the certification of large quantities of voting equipment, by "verifying the election rather than the [computer] system". They achieve this through a high degree of transparency, using a combination of cryptography, monitoring and auditing to ensure privacy while detecting vote tampering. Auditing information is distributed via the Web. An important aspect of the system used in AROVE-v is that it offers a specific design targeting at providing favoring the usability of the user interface for voters.

Technical arguments for using these systems have relied on the demonstrable strength of the algorithms: violations of integrity or confidentiality are detected with very high probability, if the implementations satisfy certain properties and voters and official follow certain procedures. Thus, if the election completes with very few errors detected, then it will have delivered the correct vote count with very high probability. To deploy such a system in an election, though, decision makers would need a case showing that the whole E-voting system (not only these auditing/monitoring algorithms) is sufficiently robust in presence of realistic human behaviour, including errors and re-working of procedures for both normal use and exception handling.

The AROVE-v case study is based on an E-voting approach called "Prêt-à-Voter" [20-21]. The project uses an interdisciplinary approach, combining the cryptography and proof based approach of the core design work with a probabilistic description of fault tolerance in human-machine systems, analysis of observational and statistical evidence and usability evaluation techniques.

One aspect of an assurance case may be to assess the effects of specific design decisions. In AROVE-v, a

related activity will involve comparing several design alternatives for the "Prêt-à-Voter" system for how these alternatives impact the five factors under consideration: dependability, usability, security, privacy and trust. The next section discusses an argumentation technique that we intend to use to address some of these issues.

3. Rationale Management

The design and assessment of complex socio-technical systems (e.g. via User Centred Design approaches [2]) call for multidisciplinary teams involving software engineers, human factor specialists, security specialist, graphic designers, etc. Reaching agreement in such multidisciplinary teams is not an easy task. The expertise of each participant tends to favor specific factors, e.g. security, trust, usability or privacy. Such difficulties may lead to critical problems if some factors get more attention at design time only because of a bias of expertise with the design team and not because of sound and structured arguments.

As pointed out in [13], these factors are not independent. For instance, the attribute "availability" is common to both dependability and security. In this section, we describe work that, in contrast with earlier approaches, addresses development process issues when requirements are expressed at a high level of abstraction in terms of factors. This work is influenced by the seminal work by Toulmin [14] in the late 50s on design management and argumentation.

To support the activities related to rationale management some of the present authors (at IRIT) employ a notation called Traceability, Exploration and Analysis Model (TEAM) and its supporting tool called Design Rationale Environment for Argumentation and Modeling (DREAM). Both the notation and the tool have been applied to various case studies including rationalization of interaction techniques for landing clearances in the field of Air Traffic Control [15] and graphical representation of notations [16].

Due to space constraints we only recall here the basic principles of the notation and the tool. Due to space constraints we only outline here the basic principles of the notation and the tool. The TEAM notation extends Allan MacLean's QOC [18] in order to deal with the specificities of safety critical interactive systems. QOC's advantages are mainly its simplicity and readability making it understandable for most people in a multidisciplinary design team. The DREAM tool (available at http://liihs.irit.fr/dream/) provides support for the edition of TEAM models, the analysis of models (to check whether all the options have been argued, commented, etc.) and the exploitation of models (to allow for argumentation and traceability of the models throughout the project and to allow for reuse for further projects). The DREAM tool also supports the management of teamwork and sessions, thus supporting the traceability of decisions according to the people that made them, but also according to the time those decisions were made

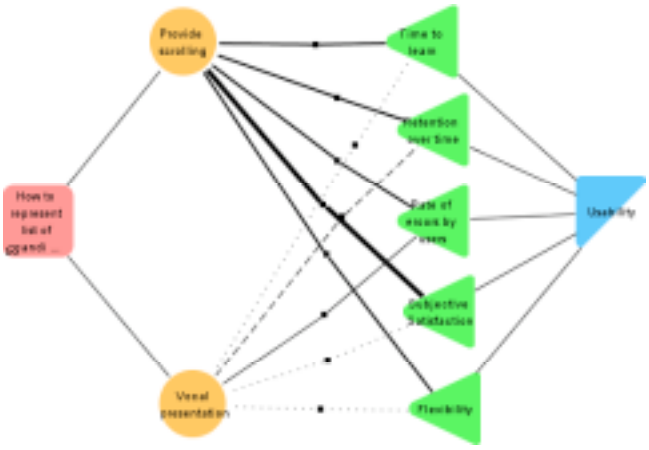


Fig. 2. Using TEAM notation to represent relations between criteria and factors in usability

Fig 2 presents a simple TEAM model for structuring argumentation about the design of the navigation model in a list of candidates for voting. Assuming that not all the candidates can be displayed in a single window, the model represents two options (circles in Fig 2.): the upper one provides scrolling facilities to the users while the lower one proposes a vocal display with navigation commands (previous and next) in that sequence. The triangles on the right hand side of the figure represent a subset of the usability criteria (time to learn, retention over time, rate of user errors, etc.; see [19] for the exhaustive list) and their connection to the usability factor. We use the terms "criteria" and "factors" as in [17]: *criteria* can be evaluated and measured while *factors* are rather abstract requirements that cannot be measured. Different types of lines in the graph represent whether a given option can support a criterion (bold line) or not (dotted line).

This notation and tool will be used to compare design alternatives for Prêt-à-Voter E-voting in terms of dependability, usability, security, privacy and trust.

4. Conclusions

Rationale management is but one of the various activities under way in the AROVE-v project. Other endeavors include:

- structuring the dependability case for the whole socio-technical system, linking overall security and dependability claims needed for E-voting to the sub-claims and evidence needed to support them. Probabilistic modelling will help in identifying critical requirements and claims, crucial mechanisms and sub-claims about them. This task will be assisted by methods and tools (also inspired by Toulmin's work [14]), such as ASCE (Assurance and Safety Case Environment) a tool developed for representing (and reasoning about) safety cases [23].
- designing and running a trial of an implementation of the "Prêt-à-Voter" E-voting system [20-21], to measure and observe relevant user behaviour and to identify potential system failure modes to inform the modelling and argumentation.

The final goal of this work is to produce recommendations for producing cases and for collecting evidence, and directions for further work in the assessment and design of complex ubiquitous socio-technical systems similar to E-voting.

Acknowledgments

This work is partly funded by the European commission via the Network of Excellence ReSIST IST-4-026764-NOE (www.resist-noe.org).

References

[1] Bloomfield, R.E. et al. Int. Working Group on Assurance Cases (for Security). IEEE Security & Privacy, vol. 4, no. 3, 2006, pp.66- 68.

[2] Norman, D & Drapper, S (Eds.). (1986) User Centred System Design. L.Erlbaum,U.S., ISBN-10: 0898597811

[3] Tolbert, C. and Mossberger, K. 2003. The effects of e-government on trust and confidence in government. In Proc. of the 2003 Annual National Conf. on Digital Government Research (Boston, MA, May 18 - 21, 2003). ACM Int. Conf. Proceeding Series, vol. 130. Digital Government Research Center, 1-7.

[4] Peters, R. M., Janssen, M., and van Engers, T. M. 2004. Measuring e-government impact: existing practices and shortcomings. In Proc. of the 6th Int. Conf. on Electronic Commerce (Delft, The Netherlands, October 25 - 27, 2004). M. Janssen, H. G. Sol, and R. W. Wagenaar, Eds. ICEC '04, vol. 60. ACM, New York, NY, 480-489.

[5] Disability. Available at: http://www.disability.gov.uk/

[6] European eAccessibility Certification. Available at: http://www.euracert.org/

[7] Official Journal of French Republic from February 11th 2005, Regulation number 2005-102.At: http://www.legifrance.gouv.fr/imagesJOE/2005/0212/joe_20050212_0036_0001.pdf

[8] Section 508: The Road to Accessibility. Available at: http://www.section508.gov/

[9] Luna-Reyes, L. F. and Gil-García, J. R. 2003. eGovernment & internet security: some technical and policy considerations. In Proc. of the 2003 Annual National Conf. on Digital Government Research (Boston, MA, May 18 - 21, 2003). ACM Int. Conf. Proceeding Series, vol. 130. Digital Government Research Center, 1-4.

[10] Hochheiser, H. 2000. Principles for privacy protection software. In Proc. of the Tenth Conf. on Computers, Freedom and Privacy: Challenging the Assumptions (Toronto, Ontario, Canada, April 04 - 07, 2000). CFP '00. ACM, New York, NY, 69-72.

[11] Dacier, M., Deswarte, Y., and Kaâniche, M. 1996. Models and tools for quantitative assessment of operational security. In information Systems Security: Facing the information Society of the 21st Century Chapman & Hall Ltd., London, UK, 177-186.

[12] Yang, L., Lu, Y., and Fu, G. 2005. Study on e-government construction. In Proc. of the 7th Int. Conf. on Electronic Commerce (Xi'an, China, August 15 - 17, 2005). ICEC '05, vol. 113. ACM, New York, NY, 542-548.

[13] Laprie, J. and Randell, B. 2004. Basic Concepts and Taxonomy of Dependable and Secure Computing. IEEE Trans. Dependable Secur. Comput. 1, 1 (Jan. 2004), 11-33

[14] Toulmin, S.E. (1958) The Uses of Argument. Cambridge: Cambridge University Press.

[15] Lacaze X., Palanque P., Barboni E., Bastide R., Navarre D. (2006) From DREAM to Realitiy: Specificities of Interactive Systems Development with respect to Rationale Management. In: Rationale Management in Software Engineering. Allen H. Dutoit, Raymond McCall, Ivan Mistrik, Barbara Paech (Eds.), Springer-Verlag/Computer Science Editorial, p. 155-172.

[16] Palanque P. & Lacaze X. DREAM-TEAM: A Tool and a Notation Supporting Exploration of Options and Traceability of Choices for Safety Critical Interactive Systems. In Proc. of INTERACT 2007, Lecture Notes in Computer Science 4662, p. 234-250 Springer.

[17] McCall, J., Richards, P., Walters, G. (1977) Factors in Software Quality. Rome Air Development Center (RADC), RADC-TR-77-369, Vol III, November.

[18] MacLean, A., Young, R.M., Bellotti, V.M.E., Moran, T.P. (1996). Questions, Options, Criteria: Elements of Design Space Analysis. In Moran, T.P. and Carroll, J.M., eds. Design Rationale: Concepts, Techniques, and Use, Lawrence Erlbaum Associates;

[19] Shneiderman, B. (1998), Designing the User Interface, Addison-Wesley Publishing Company, USA

[20] Ryan, P. Y. A. and Schneider, S. A. Prêt à Voter with Re-encryption Mixes. In Computer Security - ESORICS 2006. 11th European Symposium on Research in Computer Security, Hamburg, Germany, September 18-20, 2006. Proc., Gollmann, D., Meier, J. and Sabelfeld, A. (eds.), pp 313-326 Lecture Notes in Computer Science, 4189 Springer-Verlag, 2006, ISBN 978-3-540-44601-9

[21] Randell, B. and Ryan, P.Y.A (2006) Voting Technologies and Trust. IEEE Security & Privacy Vol. 4, Issue 5, pp 50-56. IEEE Computer Society.

Towards Information System Security Metrics

Geraldine Vache
CNRS-LAAS; Université de Toulouse - 7, Avenue du Colonel Roche, F-31077 Toulouse, France
Email: gvache@laas.fr

Abstract

The hazards related to information system security raises the need of metrics to quantify the information system security regarding security threats. This article describes our methodology to develop quantitative metrics for information system security. The purpose is to obtain a metric which represents the probability for an information system to be successfully attacked before a given time. For this aim, we focus on two parameters of the system's environment: 1) the vulnerability life cycle marked by three events: the vulnerability discovery, the vulnerability publication and the patch publication; 2) the users' behavior which may be more or less careful about the published security patch installation.

1. Introduction

At the same time the information systems' importance increases, the number of vulnerabilities related to those information systems was multiplied by five since 2000 to exceed 8000 vulnerabilities in 2006 [1]. According to these reports, the quantitative security evaluation raises the need for prevention and forecast. Our approach aims at quantitative security evaluation considering a given vulnerability. Moreover, we focus on the analysis of vulnerability life cycle which represents an important parameter for the attack process evolution.

The rest of the article is organized as follows: the next section presents the existing work related to the information system security evaluation. The third section describes our methodology using Petri net. Finally, conclusions and perspectives are presented..

2. State of the Art

The first security metrics appear to evaluate military information system security. These first methodologies are presented in a first brief section. A second brief section presents several quantifying security approaches.

2.1. Early approaches to security metrics

The first works about information system security evaluation go back in 1985 with the publication of the *Trusted Computer Security Evaluation Criteria* [2], also called "*orange book*". This standard of the *American Department of Defense (DoD)* became the reference document of information system security evaluation. The evaluation of a system using these criteria consists in the attribution to the system of one of the seven described security levels. As a follow up, several countries proposed their own evaluation method. The most well known are the *ITSEC (Information Technology Security Evaluation Criteria)*, supported by the *European Community*, and the *CTCPEC (Canadian Trusted Computer Product Evaluation Criteria)*, supported by Canada. The *Common Criteria* is the harmonization of these standards. They have been published as ISO standards: ISO 27001 [3] and ISO 27002 [4]. These two ISO standards are the first ones of the ISO 27000 standard category, devoted to information system security. However, these approaches are qualitative security evaluation methodologies but do not enable to provide quantitative metrics. Therefore, several quantitative security evaluation approaches were developed. The most important ones are presented below.

2.2. Approaches to quantitative security metrics

In 1993, the effort is presented as a better unit than time to measure security [5], but no metric is presented. The same year, the LAAS presents a model: the privilege graph [6, 7]. The privilege graph is built on the identified vulnerabilities in the system and the privileges an attacker would obtain exploiting these vulnerabilities. The graph highlights all the paths

symbolizing the various combinations of vulnerability exploitations which an attacker can use to obtain the privileges he looks for. Each arc has a weight: the effort needed to obtain the new privileges. The metric *Mean Effort To Failure (METF)* is the weighted sum of weights of the arcs composing the attacker successful paths.

The attack graph formalization is described in [8]. Each graph state represents the attacker's privileges and the overall knowledge of the system state. Thus, each event on the system is described by a transition between two states even if this event does not change the set of attacker's privileges. The approaches to generate and reduce attack graphs are presented in [8, 9, 10].

The exploitability metric, presented in [11], is based on attack tree formalization analysis. Exploitability is evaluated for each branch of the attack tree. The final metric is a global exploitability metric considering the complete attack tree.

A metric called *Time-To-Compromise* is presented in [12] and is based on the execution of three different processes: 1) the attacker knows at least one vulnerability that leads him to obtain the required privilege and at least one exploit for this vulnerability; 2) the attacker knows at least one vulnerability that leads him to obtain the required privilege but does not know any exploit for this vulnerability and looks for one; 3) the attacker tries to identify new vulnerabilities and exploits. The first two processes are exclusive and only concern the known vulnerability exploitation. The second process is executed only if the first process failed and then if its starting conditions are not valid anymore. The third process is an infinitely loop executed concurrently to the first and the second processes. The metric *Time-To-Compromise* depends on the probability of process occurrence and the success time for each of them. The metric computation uses the number of available vulnerabilities in the analysed component and the considered attacker competence level.

Three complementary metrics are represented in [5]: 1) the basic metric characterizes the necessary access rights for a vulnerability exploitation and the impact on confidentiality, integrity and availability; 2) the temporal metric deals with the existence of an exploit and a patch for a concerned vulnerability; 3) the environmental metric characterizes the vulnerable system distribution and measures the damages of an attack on system's environment. Numerical equations enable to obtain quantitative values for these three metrics but the numerical coefficient values of these equations are not justified.

Compared to the previous approaches, our approach considers that the probability of an attack using a particular vulnerability is not constant. The probability that an attacker tries to exploit a vulnerability which was not corrected and so will do a lot of victims is high. On the contrary, the probability that an attacker tries to exploit a vulnerability having a corrective patch available for a long time and largely applied, seems lower. So the vulnerability effects and consequences on the system depend on the age of the vulnerability as it is presented in the following section.

3. Methodology adopted

Our objective is to obtain a metric that expresses the probability of system intrusion by the exploitation of a particular vulnerability. For that purpose, we target our modeling on a single vulnerability and its influence on the system's environment. Our approach is based on the analysis of a phenomenon influencing attack process: vulnerability life cycle. In this section, we first present the vulnerability life cycle and define the events separating the vulnerability evolution phases. Then, we present the user's behavior influence when he is confronted with vulnerability presence and present our modeling. Finally, we consider a modeling issue: the attack probability evolution inside a vulnerability life cycle phase.

3.1. The vulnerability life cycle

The vulnerability exploitation rate varies according to vulnerability age. Moreover, it is notable that life of an existing vulnerability is marked by events having a great influence on its exploitation rate: 1) the **vulnerability discovery**. This discovery may be made by a malicious (black hat) or non malicious (white hat) person. This may have a weak influence on attack evolution, but we consider it negligible [13]; 2) the **vulnerability publication**; 3) the **vulnerability patch publication**. Figure 1 describes qualitative evolution of the number of attacks according to the three quoted events [16]. We can notice three distinct exploitation phases. On this figure, it is considered that attack process begins immediately after the vulnerability discovery, which may not be the case. For example, Microsoft published the Slammer worm vulnerability and patch on July 24th, 2002 and the worm's appearance has been estimated on January 25th, 2003 [14]. Our modeling, further presented, takes account of

this possibility.

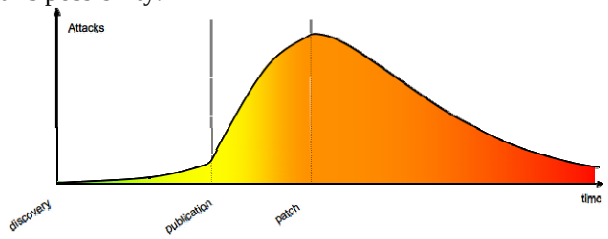


Figure 1: attack qualitative evolution considering vulnerability life cycle

3.2. Vulnerability patch installation

We have seen in the previous section that vulnerability patch publication was a significant event of vulnerability life cycle. In this section, we focus on user's installation of vulnerability patch. The user's information system update frequency may vary if this one is more or less careful and aware of the security objectives. So, we can consider the vulnerability patch installation as a random phenomenon.

The example of the Slammer worm [14] that we presented in the previous part is a good illustration of this phenomenon: the Slammer worm exploited vulnerability patch had been published six months before the worm propagation beginning. However, nearly 75.000 machines were infected by Slammer in the thirty minutes following the beginning of the epidemic. So, this phenomenon seems important enough for us to been considered in our modeling.

3.3. Modeling description

Our modeling takes several parameters into consideration: 1) the main events of vulnerability life cycle we studied in section III.A and the potential existence of an attack; 2) the user's care of security objectives; 3) the general state of the system.

We choose to model our approach by a Petri net because this formalization easily highlights system and system's environment behavior we chose to analyse. The built Petri net is represented on Figure 2.

Vulnerability life cycle phases we identified are represented by the set of places $\{V_{CREATED}, V_{DISCOVERED}, V_{PUBLISHED}, V_{PATCHED}\}$. They are separated by the events defined in section II.A and represented by the set of transitions {*discoveryV*, *publicationV*, *patchV*}. The place $\neg V_{CREATED}$ means that the vulnerability – and the component which contains it – does not exist yet. When the transition *creationV* is fired, the vulnerability has been created and may be installed and discovered.

In the place *¬Install*, the component containing the vulnerability has not been installed yet and the system is not in danger. The fire of the *InstallationV* transition means that the vulnerability is in the system. While the vulnerability has not been discovered, it could be in the system without being exploited. When the vulnerability is discovered, an exploit may be found by an attacker – modeled by the *ExploitationV* transition from the place *¬Exploit* to the place *Exploit*. Thus, the system becomes vulnerable. Three conditions are necessary to make the system vulnerable to a specific vulnerability: 1) the vulnerability was created and discovered; 2) there is an exploit for this vulnerability; 3) the vulnerability is installed in the system. As soon as these three conditions are valid, the system is considered as vulnerable. In our modeling, this is represented by the fire of one of the immediate transitions.

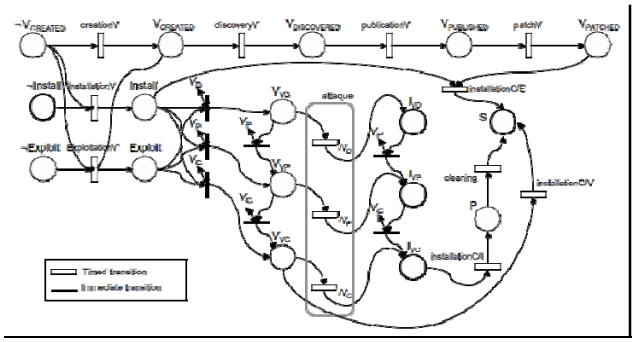


Figure 2: System and environment Petri Net modeling

In presence of the vulnerability, the system may be in four different states: 1) the **vulnerable** state, represented by the set of places $\{V_{VD}, V_{VP}, V_{VC}\}$: the vulnerability is in the system but has not been successfully exploited yet; 2) the **introduced** state, represented by the set of places $\{I_{VD}, I_{VP}, I_{VC}\}$: the vulnerability has been successfully exploited; 3) the **patched** state, represented by the place *P*: the corrective patch was installed but the damages caused by the intrusion have not been repaired yet; 4) the **secure** state, represented by the place *S*: the patch has been installed and there is no or no more damages caused by an intrusion. The states vulnerable and introduced of the system are represented by three places each. The three places $\{V_{VD}, V_{VP}, V_{VC}\}$ (respect. $\{I_{VD}, I_{VP}, I_{VC}\}$) correspond to the vulnerable (respect. introduced) state considering a particular vulnerability life cycle phase: the vulnerability has been discovered, discovered and published, or discovered, published

and patched.

The state introduced is the result of a successful attack exploiting the vulnerability against the system. This attack is symbolized by three transitions {$attack_{/VD}$, $attack_{/VP}$, $attack_{/VC}$}. Each transition represents the attack event considering the vulnerability life cycle phase. This is the probability of being in one of these three places that we try to quantify.

The system may be in the patched state or secure state only if the vulnerability patch was published and installed. The system could be in patched state if the *installationP/I* transition is fired. It indicates that the vulnerability has been patched but the damages caused by the attacker have not been repaired. The system could be in the secure state without having been introduced, by firing *intallationP/* $\bar{E}$ or *installationP/V* transitions. The system may also be in the secure state by firing the *cleaning* transition from the patched state.
Our approach describes an attack process using certain vulnerability. We have noticed in the section III.B that the vulnerability patch installation was a probabilistic event. The other events described in our modeling are deterministic events. However, considering the context of our modeling, we can represent them as probabilistic events. As an example, let us consider the attack phenomenon. If a vulnerability was discovered and there is an existing exploit, this vulnerability will be exploit. But in our context, we focus on attack phenomenon restricted to one system, so we can represent the attack process as a probabilistic event in our modeling as we do not know when the attack will take place.

4. Conclusion and perspectives

Our approach is based on system's environment evolution considering any attack evolution in the time space. Our modeling aims at obtaining the probability of intrusion according to a given vulnerability. For this purpose, we need to determine the probability distribution for each event and determine if the Markov chain's properties are valid. We plan to use the Möbius modeling tool which is developed by the PERFORM research group of the Center of Reliable and Hard Performance Computing at University of Illinois at Urbana Champaign. Möbius is a software tool for modeling the behavior of complex systems which are modeled with any probability distribution (e.g. Markov context). Once our modeling validated for several real vulnerability data, we plan to categorize vulnerabilities. This would enable a model parameter generalization for each vulnerability's category. This approach would enable us to apply our methodology to undiscovered vulnerabilities.

5. References

[1] www.cert.org
[2] *U.S. Department of Defence Trusted Computer Security Evaluation Criteria*, 5200-28-STD, 1985
[3] ISO/IEC 27001:2005, *Requirements for Information security management systems*, 2005
[4] ISO/IEC 27002:2005, *Code of practice for information security management*, 2005
[5] B. Littlewood, S. Brocklehurst, N. Fenton, P. Mellor, S. Page, D. Wright, *Toward operational measures of computer security*, Journal of Computer Security, vol 2, pp 221-229, 1993
[6] M. Dacier, *Vers une évaluation quantitative de la sécurité*, PhD, LAAS Report n° 94488, 1994
[7] M. Dacier, Y. Deswarte & M. Kaâniche. *Quantitative assessment of operational security : models and tools*, 12th IFIP Information System Security Conference, Samos, Greece, May 21-24, Chapman & Hall, 1996
[8] O. M. Sheyner. *Scenario graphs and attack graphs*, Thesis of School of Computer Science, Computer Science department, Carnegie Mellon University, Pittsburgh, PA, 2004
[9] S. Jha, O. Sheyner and J. Wing, *Two formal analyses of attack graphs*, Computer Security Foundation Workshop, 2002
[10] L.P. Swiler, C. Phillips, D. Ellis, S. Chakerian, *Computer-Attack Graph Generation Tool*, DARPA Information Survivability and Exposition II, 2001
[11] D. Balzarotti, M. Monga, S. Sicari, *Assessing the risk of using vulnerable components*, Quality of protection – Security measurements and metrics, p65-77, 2006
[12] M.A. McQueen, W.F. Boyer, M.A. Flynn, G.A. Beitel, *Time-to-compromise Model for cyber risk reduction estimation*, Quality of protection – Security measurements and metrics, p49-64, 2006
[13] E. Rescorla, *Is finding security holes a good idea ?*, IEEE Security & Privacy, Vol 3 Issue 1, pp 14-19, Jan-Feb 2005
[14] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, N. Weaver, *Inside the Slammer worm*, IEEE Security & Privacy, Vol 1 Issue 4, pp 33-39, Jul-Aug 2003
[15] P. Mell, K. Scarfone, S. Romanosky, *A complete guide to the Common Vulnerability Scoring System, version 2.0*, Forum of Incident Response and Security Team, Juin 2007
[16] N. Fishbach, *Cycle de vie d'une vulnérabilité*, Eurosec talk 2003

Towards Data Security in Affordable Data Warehouses

Marco Vieira
CISUC/DEI
Univ. of Coimbra, Portugal
mvieira@dei.uc.pt

Jorge Vieira
CISUC/DEI
Critical Software, Portugal
jvieira@criticalsoftware.com

Henrique Madeira
CISUC/DEI
Univ. of Coimbra, Portugal
henrique@dei.uc.pt

Abstract

The Data Warehouse Striping (DWS) technique is a round-robin data partitioning approach especially designed for affordable data warehousing environments based on clusters of low-cost computers running low-cost open-source software, which guaranties a nearly optimal speed up and scale up when new nodes are added to the cluster. However, low-cost software does not provide the security capabilities needed to protect critical business data. For example, most open-source Database Management Systems (DBMS) lack in providing efficient data encryption mechanisms essential to guarantee data confidentiality. This paper presents our current work on developing an architecture for affordable data warehouses with extended data protection capabilities and tolerance against nodes Denial of Service (DoS) attacks. The approach uses data encryption, spurious data, signatures, and redundancy to guarantee full data protection even when an attacker gets administrative access to one or more cluster nodes.

1. Introduction

A data warehouse (DW) is an integrated and centralized repository that offers high capabilities for data analysis and manipulation [3]. In data warehousing the data is organized according to the multidimensional model [3], which includes facts and dimensions. Facts are numeric or factual data that represent a specific business or process activity and each dimension represents a different perspective for the analysis of the facts. The multidimensional model is typically implemented as one or more star schema made of a large central fact table surrounded by several dimensional tables related to the fact table by foreign keys [3].

Data warehouses are repositories that usually contain high volumes of data integrated from several different operational sources. Thus, the data stored in a DW can range from some hundreds of Gigabytes to dozens of Terabytes. In order to properly handle high volumes of data, allowing performing complex data manipulation operations, enterprises normally use high performance systems to host the DW. The most common choice is systems that offer massive parallel processing capabilities [4], [1], as Massive Parallel Processing (MPP) systems or Symmetric MultiProcessing (SMP) systems. Due to the high price of this type of systems, some less expensive alternatives have already been proposed and implemented. One of these alternatives is the Data Warehouse Stripping (DWS) technique [2].

In a simplified view, the DWS technique consists in the distribution of the data of a data warehouse over a cluster of low-cost computers, providing near linear speedup and scale up when adding new nodes to the cluster. To achieve low-cost, the data warehouse cluster is based on open-source software and the computers can be shared with other applications (whose typically do not exploit all computational resources of the machines). However, open-source software (and Database Management Systems (DBMS) in particular) normally does not provide the full security capabilities needed to protect critical business data. Furthermore, sharing the computers with other applications increases the risk of security attacks as several users can have administrative access to the machines.

One of the main problems faced by system administrators is the protection of the data against unauthorized access or corruption due to malicious actions. In fact, due to the impressive growth of the internet, security attacks have become one vital concern in any information infrastructure. Database security arises from the need to protect from: 1) intentional unauthorized attempts to access private data, and 2) loss or corruption of critical data due to malicious actions. Other concerns include protecting against undue delays in accessing or using data, or even against malicious interferences that may cause Denial of Service (DoS).

Security is an integrative concept that includes the following properties [7]: confidentiality, authenticity, integrity, and availability. In this paper we present our current work on achieving high data security in affordable data warehouses. The goal is to endow DWS with the capabilities needed to fulfill all the data security properties, providing extended data protection capabilities and tolerance against nodes DoS attacks.

Data confidentiality is achieved by encrypting the dimensions data. Facts data is not encrypted due to performance issues (encryption in large tables is a heavy process that typically ruins the system performance [8]). Nevertheless, to improve confidentiality, facts data is obfuscated by adding spurious records to the fact tables in order to mislead the attacker. Data authenticity and integrity are guaranteed by using signatures in all records in the data warehouse and concurrent detection of malicious data modifications. Finally, data availability is achieved using replication.

The structure of the paper is as follows. Section 2 presents the DWS technique. Section 3 discusses the approach for data protection in DWS. Section 4 concludes the paper and presents the future work.

2. Using the DWS technique to build affordable data warehouses

Low cost platforms based on the DWS approach and open source technology can be employed to allow small and medium enterprises to acquire and use data warehousing technology. Our goal is thus to develop a technology that allows a dramatic reduction of the hardware, software, and administration cost when compared to traditional data warehouses based in high-end servers and proprietary software. As shown in Figure 1, our proposal is to use parallel query processing in low-cost clusters of computers running inexpensive open-source software,

In the DWS technique [2] the data of each star schema of a data warehouse is distributed over an arbitrary number of nodes having the same star schema (which is equal to the schema of the equivalent centralized version). The data of the dimension tables is replicated in each node of the cluster (i.e., each dimension has exactly the same rows in all the nodes) and the data of the fact tables is distributed over the fact tables of the several nodes using strict row-by-row round-robin partitioning or hash partitioning (see Figure 2). It is important to emphasize that the replication of dimension tables does not represent a serious overhead because usually the dimensions only represent between 1% and 5% of the space occupied by all database [3]. In the rare cases in which the star schema has a very large dimension it is possible to accommodate that dimension in the DWS cluster by using selective loading techniques [5] or encoding techniques [6].

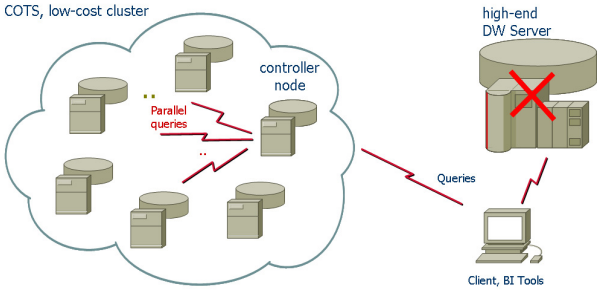


Figure 1. Architecture for low-cost DWs.

DWS data partitioning for star schemas balances the workload by all computers in the cluster, supporting parallel query processing as well as load balancing for disks and processors. The experimental results presented in [2] show that a DWS cluster can provide an almost linear speedup and scale up.

In a DWS cluster typical OLAP (On-Line Analytical Processing) queries are executed simultaneously by all the nodes available and the results are merged by the DWS middleware. As the use of a large number of inexpensive nodes increases the risk of having node failures that impair the computation of queries, DWS uses selective replication of data over the cluster nodes to guarantee full availability when nodes fail.

3. Data protection in DWS nodes

The goal of our work is to endow DWS with the mechanisms needed to assure data authenticity, integrity, confidentiality, and availability. This section presents our current research thoughts.

3.1. Assuring data authenticity and integrity

Data authenticity and integrity can be guaranteed by using signatures in all records in the DW. Each record in each table must have an associated signature that allows DWS to distinguish original data from tampered data. Obviously the signatures generation and verification must be controlled by the DWS middleware.

Record based signatures (i.e., signatures based on all columns of each record) may not be the most interesting approach. In fact, using record based signatures requires that all columns from a record are read in order to verify the signature, which may not be the best approach in terms of performance. Using one signature

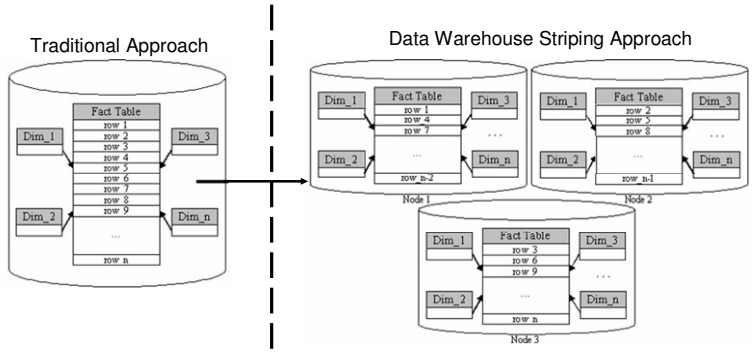


Figure 2. Data Warehouse Striping Technique.

for each column in each record is an alternative; however it brings a storage space problem that also influences performance. Our goal is to investigate the possibility of having a single signature that can be applied to validate each column individually and also to validate the entire record at once, while maintaining high-performance.

The DWS middleware has to generate the signatures when inserting or modifying data in the DW. Those signatures are used later during queries execution to guarantee that the data is authentic and has not been maliciously changed. If an authenticity or integrity problem is detected then the system must assure that that data is not used (see section 3.3).

3.2. Assuring data confidentiality

To provide communication in a secure way and avoid the access to data transferred over the network, most DBMS provide encryption mechanisms for data communication. In some cases, mechanisms that allow the encryption of the data stored in the database tables are also provided. These mechanisms are quite useful as they allow protecting the data even when someone maliciously accesses the system. However, using encryption for data storage is a heavy process that may ruin the performance of the system. In fact, previous work shows that even the encryption algorithms provided by the well-known and quite sophisticated Oracle DBMS cause very high performance degradations (representing in some cases an increase of 1000% in the response times).

Due to this performance issue, data encryption should be used only for tables with a small number of records. Tables that store large amounts of data must not be encrypted, which means that an alternative approach is required to guarantee the confidentiality of the data in those cases.

Our approach to achieve data confidentiality in DWS clusters consists in encrypting the dimensions data, which typically resides in small size tables (usually the dimensions only represent between 1% and 5% of the space occupied by all database). One of the research efforts needed is to identify the data encryption mechanisms to be used (the ones that offer higher security with lower performance impact). An important aspect is that primary keys and foreign keys in dimensions do not need to be encrypted as they are typically filled with synthetic values. The combination of encryption with encoding techniques [6] to reduce the size of large dimensions is also going to be explored.

Facts data cannot be encrypted due to performance issues. In fact, as a fact table may store several gigabytes of data, the use of encryption would ruin the queries response times (although some critical fact columns may have to be encrypted despite of the performance problems). However, as in DWS the facts data is fragmented across the cluster nodes using round-robin or hashing distribution, the confidentiality issue is minimized (additionally facts data cannot be easily related to the dimensions as these will be encrypted). In fact, when an attacker gets admittance to one or more nodes he only has access to portions of the data that are typically meaningless without their counterparts. However, there may be some cases in which the attacker may retrieve some critical facts information using solely the data stored in the attacked node (it obviously depends on how the data distribution is performed). This way, to improve confidentiality, facts data will be obfuscated by adding spurious records to the fact tables in order to mislead the attacker. The original and spurious records can be distinguished based on the use of different data signatures (which must also guarantee the authenticity of the data even for the spurious records).

Note that, adding spurious records to the fact tables may cause a performance overhead. However, we believe that this overhead will be considerably smaller than the overhead caused by data encryption. Further investigation is obviously needed to identify the number of extra records to include in each table and how to vary the facts values to effectively mislead the attacker.

To improve the privacy of the data, table names and column names (among other database objects) may also be encrypted. This difficult the task of the attacker as it becomes more difficult to understand the meaning of each table and column. Obviously the DWS middleware must be able to translate the user queries that use the original names into queries using the encrypted table and column names.

3.3. Assuring data availability

A DWS cluster is typically based on inexpensive nodes. However, the use of a large number of inexpensive nodes increases the risk of having node failures that impair the computation of queries. This way, DWS includes a redundancy mechanism, named RAIN (Redundant Array of Inexpensive Nodes), able to tolerate failures of several cluster nodes (the number of node failures tolerated depends on the configuration used). The RAIN technique is based on the selective replication of data and comprises two redundancy schemes: simple redundancy (RAIN-0) and striped redundancy (RAIN-S). The simple redundancy approach consists of replicating the facts data from each node in other nodes of the cluster. The stripped replication is an evolution of the simple replication where the facts data from each node is randomly distributed in N-1 sub-partitions (where N is the number of nodes) and each sub-partition is replicated in at least one of the other nodes.

Although the RAIN mechanism was initially designed to tolerate node failures it can also be used to guarantee data availability in the presence of data attacks. In fact, the RAIN mechanism is already able to automatically identify unavailable nodes and redirect the queries to the alternative replicas. Obviously, this allows DWS to tolerate attacks that cause nodes DoS.

For the attacks that affect the data integrity (i.e., where the attacker maliciously modifies the data), we need to add a mechanism able to concurrently detect malicious data modifications. This mechanism, based on the data signatures used to guarantee authenticity and integrity (see section 3.1), must automatically disable nodes when malicious modifications are detected. The RAIN mechanism can then be used to redirect the queries to the available replicas. An important aspect is that the detection mechanism must be built in such way that can not be tampered by the attacker (e.g., triggers are not a good approach as the attacker can easily deactivate them).

4. Conclusion and future work

In this paper we discuss our current research work on achieving high data security in DWS clusters. Data confidentiality is achieved by encrypting the dimensions data. Facts data is obfuscated by adding spurious records to the fact tables to mislead the attacker. Data authenticity and integrity are guaranteed by using signatures in all records in the DW and concurrent detection of malicious data modifications. Finally, data availability is achieved by using data replication.

Several interesting aspects have to be researched to achieve our goals. In fact, we need to study different encryption mechanisms in order to select one (or several) that provide high confidentiality with low performance costs. Facts obfuscation by using superfluous records is also an interesting challenge. Some aspects that have to be investigated include the number of extra records to include in each table and how to vary the facts values to effectively mislead the attacker. Concerning signatures we need to identify the best method for the DWS context. A mechanism to concurrently detect unauthorized data modifications based on signatures has also to be defined.

Obviously one of the main research efforts to be undertaken is related to the evaluation of the approach in terms of the impact in the DWS cluster performance. In fact, we need to compare the baseline performance (DWS without security mechanisms) with the performance achieved when using our approach and with the performance achieved when using data encryption in the fact tables.

5. References

[1] Agosta, L., “Data Warehousing Lessons Learned: SMP or MPP for Data Warehousing”, DM Review Magazine, 2002.

[2] Bernardino, J., Madeira, H., "A New Technique to Speedup Queries in Data Warehousing", ABDIS-DASFA, Symposium on Advances in DB and Information Systems, Prague, 2001.

[3] Kimball, R., Ross, M., “The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling (Second Edition)”, Ed. J. Wiley & Sons, Inc, ISBN: 0471200247, 2002.

[4] Sun Microsystems, “Data Warehousing Performance with SMP and MPP Architectures”, White Paper, 1998.

[5] Costa, M., Madeira, H. , "Handling big dimensions in distributed data warehouses using the DWS technique", ACM Seventh Intl Workshop on Data Warehousing and OLAP, DOLAP 2004, Washington, D.C., USA, 2004.

[6] Vieira, J., Bernardino, J., Madeira, H. , "Efficient compression of text attributes of data warehouse dimensions", 7th Intl Conference on Data Warehousing and Knowledge Discovery - DaWak, Copenhagen, Denmark, 2005.

[7] C. Cachin et al, “Reference model and use cases, MAFTIA deliverable D1”, MAFTIA Project IST-1999-11583, 2000.

[8] Vieira, M., Madeira, H., "Towards a security benchmark for Database Management Systems", Intl Conf. on Dependable Systems and Networks, DSN2005, Japan, 2005.

Dependability and Security: Thesauri Creation and Clustering Experiments – an Overview and an Outlook

Gintarė Grigonytė [1], Oliver Čulo[2], Algirdas Avižienis [1], Rūta Marcinkevičienė[1]
Vytautas Magnus University[1], Institute for Applied Information Research[2]
g.grigonyte@hmf.vdu.lt, culo@iai.uni-sb.de, aviz@adm.vdu.lt, ruta@hmf.vdu.lt

Abstract

We are presenting one part of the ongoing research in the FP6 project ReSIST NoE[1] concerning the creation of a thesaurus of Dependability and Security domain and the results of document clustering experiments for enriching the Resilience Knowledge Base. We explain the principles of the thesaurus building process and resume results achieved in documents clustering experiments. The results of document clustering experiments contributed in enriching RKB.

1. Introduction

The growing complexity of the Dependability and Security field makes it imperative to build a thesaurus and an ontology of concepts that relate the multiple terminologies. A taxonomy of dependability and security that evolved over the past 25 years within the IFIP Working Group 10.4 and IEEE Computer Society Technical Committee on Fault-Tolerant Computing [1] served as the starting point for this research.

For building the thesaurus of Dependability and Security we have chosen a corpus linguistic approach [2].

2. The corpus

The corpus of text used in this research is composed of nearly 2000 papers presented at the 29 annual International Symposia on Fault-Tolerant Computing (1971-1999) and at their successors, the seven International Conferences on Dependable Systems and Networks (2000-2006). Since the abstracts of the articles carry the essence of information, only abstracts of articles were used to compile the corpus. There are 181.548 running words in the corpus.

[1] http://www.resist-noe.org

3. Building the thesaurus

Our approach to building the thesaurus is based on linguistic pattern matching for automatic terminology extraction and IDF measurement for assessment of the term informativity.

The process of thesaurus learning starts with linguistic analysis of document abstracts set (see Figure 1).

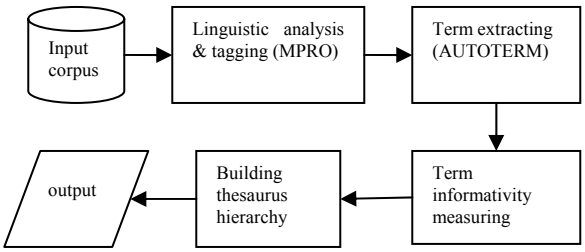


Figure 1. The process of building thesaurus

We use MPRO [3] – the program for linguistic analysis developed at IAI[2]. The output of the linguistic analysis is a flat phrase marking which enables term candidate marking based on pre-defined linguistic patterns. For this, we use an adaption of the terminology extraction tool AUTOTERM [4]. After this stage we obtained a list of 6818 term candidates [5].

In order to judge the discriminatory power of term candidates, following criteria were considered to be important: a term can not be too general and term can not be too specialised. To follow these requirements measurement of inverse document frequency (IDF) was chosen. From the IDF values obtained, we set thresholds stating that terms with $2 > idf(t) > 7$ should not be part of the candidate set. It helped to reduce term candidate number from 6,818 to 5,710, ruling out singletons and too general terms like "system" which appear in almost every document.

To create a preliminary hierarchy from general to more special terms we used a simple method: non-compound terms are top level hierarchy nodes; for a term t_x with n

[2] http://www.iai.uni-sb.de

compound parts, we look up whether there is a term t_y consisting of the *n-1* rightmost term parts; if so, the term t_x becomes a subterm of t_y.
The last step has resulted in a reasonably refined, hierarchically structured and ranked thesaurus with 5.710 terms.

4. Document clustering experiments

We have previously discussed a method for extracting terminology from a corpus based on linguistic analysis. The extracted terms were collected into a terminology database. Based on this database, each document was *indexed*. The index contains the list of terms for each document and a term weight ranging from 0 (appears, but not important for this document) to 100 (very important for this document).
In order to compute the similarity between two documents we built a representation of a vector for each document. To parallelise two vectors *a* and *b,* we use the weights of the documents descriptors as keys to the vector dimensions, and sort them alphabetically.
Our implemented clustering method is *correlation clustering* as described in [6]. Correlation clustering takes into account correlations between one document and all other documents at once.
For two documents d_i and d_j, the correlation is computed as follows:

$$corr_{ext}(d_i, d_j) = corr\left(\begin{pmatrix} corr_{1i} \\ corr_{2i} \\ \dots \\ corr_{ij} = 0 \\ \dots \\ corr_{ji} = 0 \\ \dots \\ corr_{ni} \end{pmatrix}, \begin{pmatrix} corr_{1j} \\ corr_{2j} \\ \dots \\ corr_{ji} = 0 \\ \dots \\ corr_{jj} = 0 \\ \dots \\ corr_{nj} \end{pmatrix}\right)$$

The advantage of this extended correlation measure $corr_{ext}$ is that the similarity values obtained expose a stronger contrast between candidates for similar and dissimilar documents than standard methods.

5. Conclusions

We have employed corpus linguistic approaches and NLP tools to build the thesaurus of Dependability and Security. The current version of thesaurus contains 5710 hierarchically ordered terms.

Using the described methods for document clustering we have produced 178 clusters. Out of almost 2500 documents, 200 documents remain ungrouped [7]. Manual inspection of the cluster suggests a more than acceptable quality of the results.

6. Further work

The goal of the second phase of the project will be to explore automatic methods for creating ontologies. The following tasks will be performed during 2008:

Term extraction. The term extraction process will be performed for the technical documents generated by the ReSIST NoE project up to the present time. The additional terms obtained will be used to enrich the thesaurus generated from the FTCS+DSN corpus.

Classification experiment. The results of a currently ongoing manual classification procedure will be used to implement an automatic classification experiment for the FTCS+DSN corpus. The results will be validated by D&S domain experts and used to build the upper nodes of the thesaurus.

Thesaurus enrichment. Synonymy and other relations will be determined and used to enrich the thesaurus of the D&S domain. The hierarchical thesaurus and the clustering results will be used in experiments on computational techniques for creating ontologies. Comparison of the results of two or more approaches will serve to validate the results.

Ontology representation. A logic representation of the D&S domain ontology will be created using the enriched thesaurus. Separately, the ALRL taxonomy will be manually restructured by introducing the logic representation of other relations.

Visual analysis. The interactive visualization and analysis techniques developed at Ulm university will be employed to represent and to analyze the D&S domain ontology.

7. Acknowledgments

ReSIST is a three-year European Network of Excellence project with 18 partners that integrates researchers in the fields of dependability, security, and human factors. One important topic of research that is a part of WP1 of ReSIST is the building of a thesaurus and an ontology of dependability and security.

The research described here has been carried out as a collaborative effort between the Center of Computational Linguistics at Vytautas Magnus University (VMU) and the Institute for Applied Information Research (IAI) associated with the University of Saarland, Germany.

As a consequence of the work having been done in 2007, a project was proposed by VMU and Ulm University to the ReSIST EB and has been granted for the year 2008. Staff members of the IAI will take part in the work as affiliate members of VMU Kaunas.

8. References

[1] Avižienis, Algirdas, Jean-Claude Laprie, Brian Randell & Carl Landwehr, "Basic concepts and taxonomy of dependable and secure computing". In: *IEEE transactions on dependable and secure computing*. IEEE Computer Society Press, vol.1, no.1, pp.11-33, 2004.

[2] Avižienis, Algirdas, Oliver Čulo, Gintare Grigonyte, Ruta Marcinkeviciene, "Building a Thesaurus and an Ontology of the Concepts of Dependability and Security", In: *DSN 2007.*

[3] Maas, Heinz Dieter. Multilinguale Textverarbeitung mit MPRO. In: *Europäische Kommunikationskybernetik heute und morgen '98*. Paderborn, 1998.

[4] Haller, Johann. AUTOTERM – automatische Terminologieextraktion Spanisch-Deutsch. In: *Multiperspektivische Fragestellungen der Translation in der Romania*, Alberto Gil/Ursula Wienen (eds.). Peter Lang Verlag, Frankfurt. 229-242, 2006.

[5] Čulo, Oliver, et al., "Building a Thesaurus of Dependability and Security: a Corpus Based Approach", in *Proc. of the 3rd Baltic Conf. nn Human Language Technologies*, Kaunas, Lithuania, 2007.

[6]Van Gael, Jurgen & Xiaojin Zhu 2007. Correlation clustering for crosslingual link detection. In: *International Joint Conference on Artificial Intelligence (IJCAI) 2007*.

[7] Gindiyeh, Mahmoud, et al, "A Document Classification tool for the Resilience Knowledge Base of the ReSIST NoE Project", in *Proc. of the 3rd Baltic Conf. on Human Language Technologies*, Kaunas, Lithuania, 2007.

Whose "Fault" Is This?
Untangling Domain Concepts in an Ontology of Resilient Computing

B. Rodriguez-Castro and H. Glaser
Dependable Systems and Software Engineering Group
School of Electronics and Computer Science
University of Southampton
Southampton, SO17 1BJ, U.K.
{br205r, hg}@ecs.soton.ac.uk

Abstract

Certain ontology domain concepts are difficult to model due to the complexity of their definition, the number of roles that they fulfill in the ontology or the different types of relationships they participate in. To assist ontologists in overcoming these challenges, a comparative analysis of two Ontology Design Patterns (ODPs) has been carried out. A terminology is introduced to describe the role and certain reusability scenarios of domain concepts in these ODPs. These findings make explicit certain potentially implicit modeling decisions previously taken in the ontology modeling field. Our contribution is illustrated with a concrete example from an ontology of resilient computing that will benefit from the outcome of this study.

1. Introduction

Ontologies have emerged as one of the key components needed for the realization of the Semantic Web vision [5]. A detailed overview of what an ontology is, including the evolution of its definition in the literature, can be found in Section 1.2 of [9]. Ontologies bring with them a broad range of development activities that can be grouped into what is called ontology engineering [9]. Within ontology engineering, this research primarily focuses on ontology modeling, more specifically on Ontology Design Patterns (ODPs) [8] and on how they can help representing complex domain concepts such us the one presented in the next section. ODPs have evolved from the notion of design pattern (defined in [8] as "archetypal solutions to design problems in a certain context") and they are justifiably receiving a significant amount of attention by ontologists due to the preceding success achieved by software design patterns in the context of software engineering [7].

In this study, two specific ODPs are examined [11][13] and a comparative analysis of both allows the introduction of a terminology which characterizes the roles and specific reusability scenarios of the domain concepts that participate in them. These findings make explicit certain potentially implicit modeling decisions previously taken in the development of ontology models.

The rest of this report is organized as follows: Section 2 introduces an example of a use case scenario in the field of resilient computing. Section 3 presents a brief overview of the main work in relation to ontology modeling and ODPs. Section 4 provides the comparative analysis of two ODPs and its contribution in defining role and reusability attributes of ontology domain concepts and finally, Section 5 covers the conclusions gathered from this endeavor and the lines open for further investigation.

2. Motivation

One of the objectives of the ReSIST (Resilience for Survivability in Information Society Technologies) project is to create a knowledge base application in the field of resilient and dependable computing [2]. The ReSIST Knowledge Base (RKB) provides an ontologically mediated web portal that enables the end-user to browse and search different type of information in the area of resilient systems: projects, people, institutions, publications, communities of practice, courses, etc. [3].

To achieve its objectives, the ReSIST KB features an ontology in the domain of resilient systems. This ontology was built using the definitions and taxonomies presented by Avizienis et al. [4]. The domain experts in ReSIST considered this document to be a valid source to cover the concepts that had to be represented in the ontology, saving the need of going through a knowledge acquisition phase during the development process.

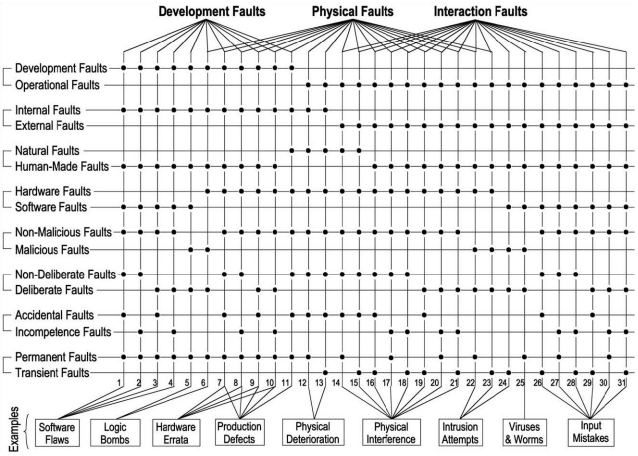


Figure 1. Matrix representation of Fault" from [2] used in the ReSIST KB ontology

Among all the ReSIST concepts, one that particularly stands out from a representational point of view is the concept of "Fault" due to: a) the complexity of its definition (Figure 1), b) the number of roles that it supports in the ontology and c) the number of relationships with other domain concepts in the same ontology. (The concept of "Fault" referred hereto and Figure 1 are explained in detail in [4]).

In the context of ReSIST, the representation of "Fault" has to fulfill the roles of a) classifying occurrences of actual faults in real world systems and b) providing a keyword index for: subjects of publications, research interest areas of projects, institutions or people, and support of resilient mechanisms.

The characteristics of role and reusability of domain concepts identified in the following sections will help to overcome the challenges caused by the multiple usages of the representation of "Fault" in ReSIST.

3. Related Research

There are several methodologies and approaches to build ontologies from scratch that address the topic of ontology conceptualization and more specifically ontology modeling. A comprehensive survey of the most relevant is provided in [6]. These methodologies (in addition of [12]), provide different levels of detail on how ontology conceptualization can be performed although they do not take into account modeling elements specific to OWL given that they are dated prior to the adoption of OWL by the W3C as the preferred ontology modeling language.

Regarding ODPs, related work that has been considered includes [8][11][13][14][1]. Different levels of ontology patterns are also discussed in [14] although in this case in the context of networked ontologies. It distinguishes three: Logical ODPs, Architectural ODPs and Content ODPs. In broad terms, Logical ODPs are equivalent to the modeling elements provided by OWL or to compositions of them. Architectural ODPs are equivalent to Logical ODPs or composition of them and characterize the structure of the ontology determining "how an ontology should look like". A basic example of Architectural ODPs would be a "taxonomy". Lastly, Content ODPs are made of Logical ODPs instances or composition of them and attempt to solve a specific domain modeling problem. The Participation, Role-Task, Design-Artifact ODPs introduced in [8] can be seen as examples of Content ODPs.

Of particular interest are the documents released by the Semantic Web Best Practices and Deployment Working Group of the W3C [1]. They provide an analysis of different approaches to a given ontology modeling problem at the level of detail required for our research and in the context of OWL as the implementation language. We note especially [11] and [13], as they will become a central piece to the outcome of our work discussed in the following section.

All the related work referred hereto; provide guidelines essential to the task of building ontologies and the usage and applicability of ODPs. However, to the best of our knowledge, they do not address approaches to model domain concepts with multiple roles or how these could be reused in the particular case of "Fault" described in ReSIST.

4. Characterizing Role and Reusability

Rector defines in [13] the concept of value partition as a modeling technique where a hierarchy of classes is used to represent features, attributes, or modifiers that describe other concepts in the ontology. According to this definition, a class in the hierarchy is partitioned by a group of subclasses if: a) the subclasses are pairwise disjoint, and b) the subclasses completely exhaust the parent class.

Conversely, Noy presents in [11] different approaches on how to use classes as property values. The situation described occurs when a hierarchy of classes is used as a subject index to annotate other domain concepts in the ontology. However, in this case there is not any restriction on the organization or characteristics that the class hierarchy may possess.

Additionally, both [11] and [13] presents a scenario, Approach 4 and Pattern 2, Variant 2 respectively, where: a) anonymous individuals are used as the value of a property for a domain concept in the ontology and b) the expressivity level of the resulting ontology is within OWL-DL.

4.1. Role of Domain Concepts

Based on this comparative analysis of [13] and [11], we introduce the following terminology to characterize the role that a given hierarchy of classes fulfills in these two ODPs examined:

Generic Class Hierarchy (GCH): refers to a set of classes organized in any hierarchical structure (e.g. a single class or a set of classes organized in a list, a tree or a directed acyclic graph) whether it conforms to the definition of value partition or not.

Domain Class Hierarchy (DCH): refers to any GCH that contains the classes corresponding to the domain concepts that the ontology is intended to represent.

Value Class Hierarchy (VCH): refers to any GCH that is used to provide anonymous individuals as values to properties for other domain concepts in the ontology.

Value Partition Class Hierarchy (VPCH): refers to a GCH that a) is a Value Class Hierarchy and b) conforms to the definition of a value partition.

Furthermore, the ontology could then be divided into two possible spaces:

Domain Concept Space (DCS): identifies the subset of the ontology model that contains all the classes that belong to a Domain Class Hierarchy.

Value Space (VS): identifies the subset of the ontology model that contains all the classes that belong to a Value Class Hierarchy or Value Partition Class Hierarchy.

It is also a good modeling practice, as mentioned in [10] and [13] to make these two spaces, DCS and VS, disjoint.

4.2. Reusability of Domain Concepts

Using the terminology above and based on the roles that class hierarchies fulfill in the ontology, the following reusability scenarios for these can be characterized:

Scenario 1: Let us consider two ontologies O_1 and O_2, with two Domain Class Hierarchies DCH_1 and DCH_2 in their Domain Concept Space respectively. It is possible to apply [11] and [13] to reuse DCH_2 from O_2 to support the role of a Value Class Hierarchy in ontology O_1. In that case:

a) A class (say C_1) of DCH_1 becomes the domain for some property (say P) in O_1.

b) A class (say C_2) of DCH_2 becomes the range for property P in O_1.

c) An anonymous individual from C_2 becomes the value for property P in O_1.

d) DCH_2 becomes part of the Value Space in O_1 and disjoint from DCH_1 in O_1.

Scenario 2: Let us consider a single ontology O_1, with two Domain Class Hierarchies DCH_{11} and DCH_{12}. It is possible to apply [11] and [13] to reuse DCH_{12} support the role of a Value Class Hierarchy for DCH_{11} in O_1. In that case:

a) A class (say C_1) of DCH_{11} becomes the domain for some property (say P) in O_1.

b) A class (say C_2) of DCH_{12} becomes the range for the property P in O_1.

c) An anonymous individual from C_2 becomes the value for property P in O_1.

d) DCH_{12} becomes part of the Value Space in O_1 causing both the DCS and the VS in O_1 to overlap.

4.3. Role and Reuse of "Fault" in ReSIST

The characteristics of role and reusability presented here helped untangling these two aspects in the modeling of the "Fault" concept in the ReSIST KB ontology. This is illustrated as follows:

a) The "Fault" domain concept is represented as a Generic Class Hierarchy resulting in a directed acyclic graph structure to accommodate the different taxonomies that define "Fault" in [4] (Figure 1).

b) The "Fault" Generic Class Hierarchy is used to represent instances of real world faults in the field of resilient and dependable systems. In that sense, it supports the role of a Domain Class Hierarchy in the Domain Concept Space of the ReSIST KB ontology.

c) The "Fault" Generic Class Hierarchy serves as a subject and keyword index for other domain concepts in the ontology. In that sense, it supports the role of a Value Class Hierarchy in the Value Space of the overall ReSIST KB ontology. For example: the "Fault" class hierarchy will supply the value for properties such as "hasResearchSubject" (associated with the concept of "Publication"), "hasResearchInterest" (associated with the concept of "Person", "Institution" or "Project") or "hasSupportFor" (associated with the concept of "Resilient Mechanism").

d) The representation of "Fault" is reused to support the role of both a Domain and Value Class Hierarchy causing the Domain Concept Space and the Value Concept Space of the ReSIST KB ontology to overlap.

This representation of "Fault" coincides with the characteristics outlined in Scenario 2 above for the Domain Class Hierarchy DCH_{12} and helped us explicitly clarify the modeling decisions to be made for this concept in the context of ReSIST.

5. Conclusions and Future Work

Similarities have been discussed between two ontology modeling design patterns that share how they use anonymous individuals to provide the value for properties in the ontology. These similarities allow expanding the notion of value partition to other structures of class hierarchies. A terminology is introduced that describes the roles of domain concepts in the ODPs considered. In the context of this terminology, certain reusability scenarios of domain concepts in ontology models were introduced. These scenarios make explicit certain potentially implicit modeling decisions previously taken in the ontology development field. Our contribution has been illustrated with the representation of the "Fault" domain concept that is part of the ontology in the knowledge base of the ReSIST project.

Current lines of future work include a study of the conceptual overlap inherent in the class hierarchies that define the "Fault" domain concept. We are working on a characterization of the different types of conceptual overlap that takes place among the multiple classification criteria that constitute the definition of "Fault". We expect that the outcome of this analysis will set the basis for the development of additional ODPs in the context of OWL to address these conceptual overlap scenarios. In addition, the area of ontology evaluation is being explored to identify opportunities for reuse of available evaluation frameworks suitable for these ODPs or the need to develop new ones tailored to our requirements.

Acknowledgements: Many people have contributed directly and indirectly to this work and we thank them all. In particular: Ian Millard, Afraz Jaffri and Madalina Croitoru. The ReSIST Network of Excellence is sponsored by the Information Society Technology (IST) priority in the EU Sixth Framework Programme (FP6) under contract number IST 4 026764 NOE.

References

[1] W3C Semantic Web Best Practices and Deployment Working Group, 2004-5. http://www.w3.org/2001/sw/BestPractices/.

[2] The ReSIST Network of Excellence, 2005. http://www.resist-noe.eu/.

[3] T. Anderson, Z. Andrews, J. Fitzgerald, B. Randell, H. Glaser, and I. Millard. The ReSIST Resilience Knowledge Base. In *DSN 2007 - The 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, June 2007.

[4] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 01(1):11–33, 2004.

[5] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, May 2001.

[6] M. Fernandez-Lopez, A. Gomez-Perez, J. Euzenat, A. Gangemi, Y. Kalfoglou, D. Pisanelli, M. Schorlemmer, G. Steve, L. Stojanovic, G. Stumme, and Y. Sure. A survey on methodologies for developing, maintaining, integration, evaluation and reengineering ontologies. OntoWeb deliverable D1.4, Universidad Politecnia de Madrid, 2002. http://www.aifb.uni-karsruhe.de/WBS/ysu/publications/OntoWeb_Del_1-4.pdf.

[7] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Professional, 1995.

[8] A. Gangemi. Ontology design patterns for semantic web content. In Y. Gil, E. Motta, V. R. Benjamins, and M. A. Musen, editors, *International Semantic Web Conference*, volume 3729 of *Lecture Notes in Computer Science*, pages 262–276. Springer, 2005.

[9] A. Gomez-Perez, O. Corcho, and M. Fernandez-Lopez. *Ontological Engineering : with examples from the areas of Knowledge Management, e-Commerce and the Semantic Web. First Edition (Advanced Information and Knowledge Processing)*. Springer, July 2004.

[10] M. Horridge, H. Knublauch, A. Rector, R. Stevens, and C. Wroe. A practical guide to building owl ontologies using the protege-owl plugin and co-ode tools edition 1.0. Technical report, The University Of Manchester, August 2004.

[11] N. F. Noy. Representing classes as property values on the semantic web. Technical Report Note 5, W3C, Semantic Web Best Practices and Deployment Working Group, 2005. http://www.w3.org/TR/swbp-classes-as-values/.

[12] N. F. Noy and D. McGuinness. Ontology development 101: A guide to creating your first ontology. Technical Report SMI-2001-0880, Stanford Medical Informatics, Stanford, 2001.

[13] A. Rector. Representing specified values in owl: "value partitions" and "value sets". Technical Report Note 17, W3C, Semantic Web Best Practices and Deployment Working Group, 2005. http://www.w3.org/TR/swbp-specified-values/.

[14] M. C. Suarez-Figueroa, S. Brockmans, A. Gangemi, A. Gomez-Perez, J. Lehmann, H. Lewen, V. Presutti, and M. Sabou. Neon modelling components. NeOn deliverable D5.1.1, Universidad Politecnica de Madrid, 2007.

Session 3B: Fast Abstracts III

Dependability Assessment and Analysis

EDCC-7

High-Speed Hardware/Software Cosimulation of Complex Systems including Fault-Injection of Detailed VHDL-Models for PC-Components

Stefan Potyra, Volkmar Sieh
Friedrich-Alexander-Universität Erlangen-Nürnberg
Departement for Computer Science 3 (Computer Architecture)
Martensstr. 3, 91058 Erlangen, Germany
Stefan.Potyra@informatik.uni-erlangen.de
Volkmar.Sieh@informatik.uni-erlangen.de

Abstract

FAUmachine is a virtual machine to simulate standard PC hardware together with an environment. FAUmachine comes with fault injection capabilities. This paper presents the current work in progress to enhance FAUmachine with a method for high-speed simulation of VHDL modelled components. It also discusses the future plans to add a mechanism, which will allow fault injection in regards to VHDL design units.

1. Introduction to FAUmachine

FAUmachine[6] is an open source virtual machine for standard PC hardware. A variety of different operating systems can be run on FAUmachine unmodified, including Windows, Linux, Dos, Free- and OpenBSD.

FAUmachine can simulate standard Intel CPUs and IDE controllers, NE2000- and Intel eepro100 network interface adapters but also peripheriphals such as networking hubs, serial terminals or modems.

One of the differences to other virtual machines like QEMU[2], VirtualBox [10], Bochs[17] or VMware[20] is, that FAUmachine can be configured on a very fine granular level. Such details include, to what memory banks a memory module is connected to, or which PCI-slot a PCI-card is inserted into.

Another difference to both QEMU and VMWare is the design goal of FAUmachine. While the aforementioned virtual machines aim at executing the simulated system as fast as possible – which is often achieved by diverging from the exact behaviour of real hardware in the case of need – FAUmachine tries to achieve a simulation as close as possible to real hardware. Additionally, FAUmachine can not only simulate a PC, but also the environment, like power switches, monitor, the power supply and even the interaction of the user.

In order to nevertheless achieve a high performance simulation, FAUmachine uses the techniques of a just-in-time compiler[8] to transform a stream of machine code instructions of the workload into a sequence of machine code instructions that can be run on the host system. If a peripheral is accessed, an instruction to call the corresponding simulation procedure is inserted. Using this technique, the overall performance of FAUmachine is about 5-20% as good as the performance of the corresponding real system.

1.1. Current fault injection capabilities of FAUmachine

Currently, fault injection inside FAUmachine is implemented inside the simulators for different components[9]. This approach has the advantage, that different components can provide specialized fault types. For example the simulator of the network card makes it possible to specify a percentile amount of package loss. To the virtual disk simulator, both persistant and transient block faults can get injected. Other currently supported fault injection mechanisms include intermittent and persistant bit-flips for both the simulated CPU and memory modules.

The corresponding C-Code of the simulated hardware tries to reflect real hardware. That makes it possible to add new fault injection capabilities, that reflect faulty behaviour of real hardware without having to rewrite large parts of FAUmachine. Nonetheless there doesn't exist a generic method to specify individual failure behaviour on a component basis. Since each simulated component needs to get adjusted, this creates an additional burden. Likewise, since the interconnection

between individual simulated components are already modelled as signals – not too unlike the signal concept in VHDL – it would be nice to have a generic method of injecting faults to such individual signals.

2. Simulation of VHDL code in conjunction with FAUmachine

A current work in progress is to make FAUmachine simulate VHDL-code in the environment of a complete system. This could be a great benefit in hardware software codesign: On the one hand, it enables rapid prototyping. When developing a piece of hardware, it is essential that it can be simulated. However current simulators, like Savant [5] or ModelSim[12] provide only a very limited scope for the simulation. It is limited to the developed hardware itself, and handwritten test-benches. Naturally, such test-benchs either will test only a small behavioural subset of the real hardware, or are extremely costly to write in the first place, if the whole state space of a VHDL component is to get tested.

Using simulator coupling to interconnect the simulation of VHDL components with other simulators[7] can help to widen the testing coverage. However this approach is based on a high level abstraction of the junctions between different simulations. Hence it cannot fulfill the need to simulate low level interaction like PCI-bus accesses, which however is often necessary to find errors in combined systems of hardware/software and is especially helpful to find errors between hardware components and the operating system drivers.

Simulating the VHDL design in the surrounding of an entire PC-system, including running an operating system can be a means to detect more errors like race conditions, which are more difficult to find. Of course successful tests in such a simulated environment do not guarantee a flawless design either; rather, these provide a different testing scenario and hence can cover different subsets of the state space.

Another beneficial effect is, that software development (e.g. of a driver) can be done in parallel to developing the hardware. While some aspects of the software can certainly get developed without any hardware at all, it is necessary to have access to a prototype of the hardware to test the large parts of the functionality of a driver. Producing such a prototype may – depending on the particular hardware – be a costly efford though.

Finally some design flaws in the hardware can be spotted when the corresponding software gets developed in parallel much easier than by analysign the hardware alone.

To achieve the simulation of VHDL inside FAUmachine, currently a compiler is being developed, which will support a large subset of the VHDL 2002 standard [1]. The output of the compiler will be an intermediate code, which can get interpreted quickly. Later, a C-backend will be added which has the intermediate code as an input language. This could provide an even better simulation speed.

3. Fine granular fault injection

Regarding the goal of designing fault tolerant systems comprised of both software and hardware, fault injection is a measure which enables developers to test their design in this regard. The current fault injection capabilities of FAUmachine however only satisfy this need partially: While FAUmachine provides methods to simulate hardware faults in the surrounding system, the simulation of VHDL design units doesn't have this feature yet. Hence a generic methods needs to get developed, to allow fault injection to such entities.

With the aim of reflecting real hardware faults as close as possible, the following requirements should get met with the developed approach:

- transparent in regard to synthesized chip layout
- emulate the failure of individual signal lines
- simulating failure of individual components

First off all, it is a great desire to have the possibility to turn on fault injection without having to change the VHDL code significantly from the original version. At the bare minimum, adding code to specify faulty behaviour should be transparent in regard to the synthesized chip layout, i.e. the adapted VHDL code must still produce an identical circuit compared to the original code.

One possible approach to achieve this is to modify solely the runtime environment used to update signals from the driving values. The *MEFISTO* project makes use of this procedure[11, 3].

A different method is to use VHDL mechanism of user-defined attributes, which only add meta-information to the design but won't affect the synthesizing result (actually, some synthesizers make use of this method so that special ways, how certain constructs should get synthesized can be defined). Another possible method, to extent the VHDL language with fault injection signals[16] could be applied a well.

Using this approach has the advantage, that faults that should get simulated can be specified directly in the design hierarchy, making it easy to see the context in which these faults will appear. Additionally, the compiler would be aware of the presence of faulty design,

and could emit different intermediate code in case fault injection is switched on or off.

However estimating fault rates of a hardware component is highly dependent on the synthesized structures: In case different optimization goals or different hardware libraries are to be used when synthesizing a design unit, vastly different gate logic can be the result. This can lead to big variances in respect to fault tolerance[15]. To alleviate this, the following method could be applied: Instead of directly compiling the VHDL code of the programmer, tools like Synopsis[18] can be used to generate VHDL code of the synthesized layout. In the hardware specific libraries that resemble different gate level logic, faulty behaviour gets specified using the aforementioned approach. Both the libraries together with the generated VHDL code get compiled by the VHDL compiler and will be used as an input for the simulation.

The disadvantage of this approach is that the resulting simulation speed of the synthesized VHDL is likely to be much lower compared to the original VHDL code, due to the highly concurrent nature of the synthesized code. Using FAUmachines abilities to perform automated tests, which will be discussed in the next section, this doesn't seem to big hindrance though: While it's essential to achieve high simulation performance in order to allow interactive tests, there obviously isn't a great urge to be able to interactively use simulated hardware which is affected by injected faults.

4. Automated testing

FAUmachine already provides the means to setup and run automated tests, including the specification of user behaviour and the specification of faults to be injected[14, 19, 4, 9]. For this, VHDL scripts define the stimuli for the interacting components on a very fine granular manner. There also exists a programming language from which behavioural VHDL scripts can get generated, which helps to create sequential test sequences in a straigtforward manner[13]. Currently, the VHDL interpreter *expect* will take care for the interaction of the test stimuli with the simulated components. To achieve this, common interfaces between the simulated components are already a part of FAUmachine. These interfaces are mapped to VHDL signals and/or VHDL procedure calls inside of *expect*. Reusing these existing methods to enable the VHDL compiler to run automated tests seems a worthwhile goal: Automated tests can then be used to do regression tests during the development process of hardware/software systems. Running such tests both with and without fault injection, will provide the additional benefit of to spot regressions in relation to fault tolerance as well.

5. Conclusion

This paper presented the work in progress to develop methods, which allow the high-speed cosimulation of complex hardware/software systems using FAUmachine. Enriched with the capability to allow fault injection to both the simulated hardware components of FAUmachine's virtual machine and the VHDL design units, it will enable developers to design and test fault tolerant hardware/software systems.

References

[1] I. S. 1076-2002. IEEE Standard VHDL language reference manual. Product No.: SH94983-TBR, 2002.

[2] F. Bellard. Qemu, a fast and portable dynamic translator. In *ATEC'05: Proceedings of the USENIX Annual Technical Conference 2005 on USENIX Annual Technical Conference*, pages 41–46, Berkeley, CA, USA, 2005. USENIX Association.

[3] J. Boue, P. Petillon, and Y. Crouzet. Mefisto-l: a vhdl-based fault injection tool for the experimental assessment of fault tolerance. *Fault-Tolerant Computing, 1998. Digest of Papers. Twenty-Eighth Annual International Symposium on*, pages 168–173, 23-25 Jun 1998.

[4] K. Buchacker, M. Dal Cin, H. Höxer, R. Karch, V. Sieh, and O. Tschäche. Reproducible dependability benchmarking experiments based on unambiguous benchmark setup descriptions. In *Proceedings of the International Conference on Dependable Systems and Networks*, pages 469–478, 2003.

[5] Experimental Computer Laboratory, University of Cincinnatti. Savant: VHDL analysis tool. URL: http://www.ececs.uc.edu/∼paw/savant/, 2003.

[6] FAUmachine Team. FAUmachine. URL: http://www.FAUmachine.org/, 2003–2007.

[7] U. Hatnik and S. Altmann. Using modelsim, matlab/simulink and ns for simulation of distributed systems. In *PARELEC '04: Proceedings of the international conference on Parallel Computing in Electrical Engineering*, pages 114–119, Washington, DC, USA, 2004. IEEE Computer Society.

[8] H. Höxer, M. Waitz, and V. Sieh. Advanced virtualization techniques for FAUmachine. In R. Spenneberg, editor, *11th International Linux System Technology Conference, Erlangen, Germany, September 7-10, 2004*, pages 1–12, 2004.

[9] H. Höxer, M. Waitz, and V. Sieh. Fast Simulation of Stuck-At and Coupling Memory Faults Using FAUmachine. In *Supplement to Proc. HASE 2005: International Symposium on High Assurance Systems Engineering (Ninth IEEE International Symposium on High Assurance Systems Engineering Heidelberg, Germany 12-14 October 2005)*, pages 1–2, Oct. 2005.

[10] innotek GmbH. VirtualBox. URL: http://www.virtualbox.org/, 2008.

[11] E. Jenn, J. Arlat, M. Rimen, J. Ohlsson, and J. Karlsson. Fault injection into VHDL models: The MEFISTO tool. In *Proceedings of the 24th IEEE International Symposium on Fault Tolerant Computing*, pages 66–75, 1994.

[12] Mentor Graphics Corp. Modelsim – a comprehensive simulation and debug environment for asic and fpga design. URL: http://www.model.com, 2007.

[13] S. Potyra, V. Sieh, and M. Dal Cin. Evaluating fault-tolerant system designs using faumachine. In *EFTS '07: Proceedings of the 2007 workshop on Engineering fault tolerant systems*, page 9, New York, NY, USA, 2007. ACM.

[14] V. Sieh and K. Buchacker. Testing the fault-tolerance of networked systems. In U. Brinkschulte, K.-E. Grosspietsch, C. Hochberger, and E. W. Mayr, editors, *International Conference on Architecture of Computing Systems ARCS 2002, Workshop Proceedings*, pages 37–46, 2002.

[15] V. Sieh, O. Tschäche, and F. Balbach. Comparing different fault models using VERIFY. In *6th Conference on Dependable Computing for Critical Applications*, pages 59–76, 1997.

[16] V. Sieh, O. Tschäche, and F. Balbach. VERIFY: Evaluation of reliability using VHDL-models with integrated fault descriptions. In *Proceedings of the 27th IEEE International Symposium on Fault Tolerant Computing*, pages 32–36, 1997.

[17] Source Forge. Bochs IA-32 Emulator Project. URL: http://bochs.sourceforge.org/, 2001.

[18] Synopsis Inc. Synopsis. URL: http://www.synopsys.com/, 2008.

[19] O. Tschäche. Dependability benchmarking of linux based systems. In *Proceedings Informatik 2003, Beiträge des Schwerpunkts Sicherheit - Schutz und Zuverlässigkeit, english*, pages 237–248, 2003.

[20] VMware Inc. VMware. URL: http://www.vmware.com/, 2001.

Safety Concerns for Tool Use in the Design of Complex Electronic Hardware

Andrew J. Kornecki
Dept. of Computer & Software Eng.
Embry-Riddle Aeronautical Univ.
Daytona Beach, FL 32114
kornecka@erau.edu

Janusz Zalewski
Dept. of Computer Science
Florida Gulf Coast University
Ft. Myers, FL 33965
zalewski@fgcu.edu

Brian Butka
Dept. of Electrical & Systems Eng.
Embry-Riddle Aeronautical Univ.
Daytona Beach, FL 32114
butkab@erau.edu

Abstract

Modern electronic devices increasingly use dedicated hardware to process the growing amounts of data to control their operation. These complex programmable electronic devices are developed by writing code in a hardware description language used to create logic designs. Most of these devices, such as FPGA, can be configured to implement a particular design by downloading a sequence of bits. In that sense, a circuit implemented on an FPGA is literally software. However, treating circuits as "hardware" poses problems in system development and certification. The objective of this paper is to discuss the need for appropriate treatment of software processes in the development of complex electronic hardware, in a view of certification. We focus on software tools for hardware development in avionics and aerospace systems, and describe a method and procedures to evaluate a tool with respect to real-time constraints, for the purpose of qualification.

1. Introduction

Modern electronic systems use not only increasing numbers of microcomputers and microprocessors but also dedicated hardware to process the growing amounts of data needed to control systems operation and monitor their status. These complex programmable electronic devices are not only programmed using conventional languages but also are developed by writing code in a hardware description language used to create logic designs. Software tools are used to simulate the logic, synthesize the circuit, and create the placement and routing for the electronic elements and their connections in preparation for the final implementation, i.e., programming the logic devices, which used to be conventionally called "burning into the logic."

The primary devices in this category include equipment based on PLD (Programmable Logic Devices), FPGA (Field Programmable Gate Arrays), ASIC (Application Specific Integrated Circuits), and similar circuits used as components of programmable electronic hardware. Often the circuit includes dedicated processors whose Intellectual Property (IP) is made into the final product silicon.

It is interesting to note that creation of the complex circuits is currently not considered a software activity and it is left to hardware specialists. However, the hardware/system description languages, such as VHDL, Verilog, or System C are basically computer languages with their own syntax and semantics. The development of hardware relies significantly on the quality of tools, which translate the software artifacts from one form to another. Both software and hardware use extensively very complex tools, i.e., integrated programming environments taking the project from its conceptual stage into the final product. It is the contention of the authors that the development of systems containing both software and hardware components needs to be done in a unified manner.

2. Concerns for Certification of Complex Electronic Hardware

The critical issue in the development of dependable safety-critical systems is the close relation between what used to be termed under separate categories: software vs. hardware. Software application designers focusing on development of programs to run on microprocessors, often overlook programmable logic devices and most popular Field Programmable Logic Arrays (FPGA). FPGA is a prefabricated integrated circuit that can be configured to implement a particular design by downloading a sequence of bits. In that sense, a circuit implemented on an FPGA is literally software. However, circuit designers are still known as hardware specialists, and the algorithms ported to circuits are still known as hardware algorithms. Treating circuits as "hardware" poses problems in computing system development, especially for embedded systems [1].

Of special concern is the fact that using both software and hardware in creation of dependable, often safety-critical systems requires meeting certain government regulations and engineering standards. For example, in the development of airborne systems installed on civilian aircraft, the software aspects of certification are guided by RTCA DO-178B "Software Considerations in Airborne Systems and Equipment Certification", which defines the processes and artifacts to meet the approval objectives.

On the other hand, RTCA DO-254, "Design Assurance Guidance for Airborne Electronic Hardware", provides details on the processes that must be followed in assessment of hardware components. In particular, the document provides certification information on project conception,

planning, design, implementation, testing, and verification. Both RTCA documents address the issue of qualification of the tools used for creation of an airborne system in a slightly different way. It is thus conceivable that a system of specific level of safety assurance (defined by the categories from A to D, i.e., from the most to the least critical) will receive different scrutiny depending whether it is implemented in software or in hardware.

Certification issues in aerospace applications are also becoming increasingly evident and important [2]. For example, decision software installed in unmanned autonomous systems (UAS) needs to be both reliable and safe. However, trusting decisions made by autonomous control software may require new methods and processes to guarantee safety. The difficulty lies in determining how these intelligent systems will operate in a dynamic environment and with little or no human oversight. UAS autonomous control requires significant technology advancement, since it does replace years of training for human operators and their decision-making abilities. Neglecting autonomous control certification research today will dramatically increase cost for future users [3].

The concern is that a designer can freely choose to implement a task in a mix of hardware and software, but the FAA regulations and the software tools struggle handling this type of a design. The objective of this paper is to discuss the need for proper treatment of software processes in the development of complex electronic hardware, in a view of the need for unification of software and hardware development for of certification. In the previous work [4], we applied the criteria for assessment of software development tools for safety-critical real-time systems. In this paper, we focus on software tools for development of complex electronic hardware, and describe a method and procedures to evaluate a tool with respect to real-time constraints for the purpose of qualification.

3. Typical Development Process for an FPGS Based System

A typical design, verification and validation flow for an FPGA based avionic system is shown in Figure 1. DO-254B requires that all verification of Level A and B functions should be independent. For level C and lower, independence is needed only at the design hierarchy where it is verified against the requirements. It is not the intent of independence to require someone other than the designer to execute the tests once they are evaluated or developed with independence. However, the results may still need to be reviewed independently to confirm proper procedures were followed and that the results verify that the requirements were met. Verification process objectives may be satisfied through a combination of reviews, analyses, and the development and execution of tests. The verification activities, such as test, simulation, prototyping, analyses and reviews should demonstrate compliance with the requirements. Since the complex electronic devices can be considered on various hierarchy levels (transistors, gates, circuits, components), it is not intended that requirements should be verified at every hierarchical level.

Many companies choose to generate internal verification and validation testbenches to provide rapid feedback and better assure design correctness. Figure 2 shows the design flow for a company utilizing internal verification techniques. The process begins with using the system requirements document to generate the hardware design specification. The design group will then implement the hardware design specification by writing Register Transfer Language (RTL) source code. RTL source code is typically written in using the Verilog or VHDL language. A separate group will use the system requirements document to generate the test plan and create a testbench. The test plan will typically consist of one or more of the following:

- Directed Tests generated to directly verify each requirement in the hardware systems document. DO-254 explicitly requires these tests.
- Constrained Random Tests consist of test vectors generated by defining a space of all legal system inputs and generating random sequences of legal inputs. This method often finds bugs that no amount of directed test does.
- Formal Proofs of correctness where the system is proven to generate the correct output for all legal inputs using mathematical and logical theorems.
- Error Condition Tests consist of applying illegal inputs to the system and verifying that system can recover from unexpected/illegal inputs in a safe manner.

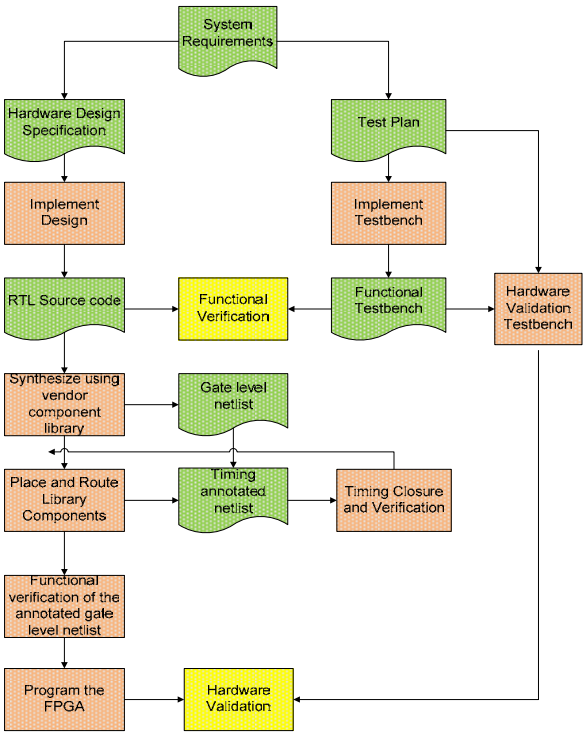


Figure 1. Typical design, verification and validation flow for an FPGA based avionic system.

The functional verification testbench in this technique typically requires many hours of simulation time to run. A typical procedure is that every evening all of the designs are checked into the design repository and functional verification is run on the most recent iteration of the design. Next the design group gets a report including every test that failed and the exact failure that occurred. When the functional verification is first being run, thousands of errors are often generated. The two groups must work together to identify whether the problem is in the design or in the testbench. Although tedious, this is often one of the most important tasks to be performed since it is common to find that the design and verification groups interpret the same specification in different ways. Clearing up errors in the system specification or its interpretation as early as possible can save many months of wasted design effort.

Once the design passes all of the functional tests it is synthesized into the hardware using the vendor's hardware library. The output of this step is called a gate level net which describes the design in terms of actual hardware available on the target device. For any given RTL source code there can be many possible implementations. The designer guides the tool in choosing the implementation by providing constraints on the design. The most common constraints are timing constraints which specify the acceptable delays on signal paths as well as register setup and hold constraints. Additional constraints can be area constraints where the amount of FPGA resources (or area) consumed are restricted, or such constraints as power dissipation.

The place and route tool implements the design in hardware and generates timing, area and power estimates for all signals. These estimates are then compared to the timing, area, and power constraints. If the current iteration does not meet the specification, the place and route tool attempts to meet the constraints by changing the physical location of the components and by changing the actual hardware implementation of the RTL source code. Although there are often multiple constraints involved, the most common problems are timing problems and this phase of the design is called timing closure. Timing closure of a large design will typically require several weeks of effort. It is possible for a timing problem to be so severe that the system architecture needs to be modified. In this case the hardware requirements will be modified, new RTL is generated, function verification performed again and then timing closure attempted using the new gate level netlist.

After the design meets timing closure the final gate level netlist is annotated with accurate timing information and functional verification is performed again using the best simulation model of the hardware. Assuming the final functional verification passes, the bitstream data for the FPGA is generated and downloaded. At this point the FPGA hardware is validated using laboratory equipment. The validation tests typically include all the test vectors used in the functional verification as well as vectors intended to check problems that cannot be easily seen in simulation.

The procedure presented above is consistent with the DO-254 guidelines that say: "as the complexity of the hardware design increases, it is advantageous to make use of computerized tools, such as simulation to verify requirements and implementation of the design." The issue is that it is a software program (written in an RTL) processed and analyzed by other software (tool) running on a general purpose workstation with conventional operating system (such as Windows) responsible for creation of hardware. The correctness and dependability of a software tool is critical to the effective and efficient creation of modern hardware.

4. Focus on Safety Issues in the Complex Hardware Development

The case studies are being developed based on the expressed concerns of the scientific and industrial communities with reference to complex electronic hardware (CEH) tools, such as those used for FPGA development. Largely these concerns are with relevance to development of safety-critical and real-time systems [4].

These case studies are geared towards qualifying the CEH tools. In an attempt to qualify the tools, the tools are going to be used with worst case scenarios along with least likely uses in order to test the bounds of the tools capability. The concept is that the black box design entered into the tool shall have a one to one mapping trace to the black box operation that is finally implemented. This however shall be done in a design independent fashion. In order to facilitate the design independence, case studies are developed as small very focused experiments that are used to discover specific attributes of a tool. This method is preferred rather that elaborate designs, due to the fact that often broken links in tracing from design to implementation are due to a flaw in the design. The individual case studies specific focus is as follows:

1. Real timing – Determination of timing based on synthesis redesign. Also used to determine the applied safety margin to the reported timings.
2. Power constraints – Systematic way of determining if place and route functions are effective.
3. Undefined I/O status – Test to determine how the tool uses the I/O not defined by the user.
4. Simulation error – Behavioral simulation results vs. actual implemented behavior; to determine accuracy and reliability of the simulation portion of the tools.
5. Faulty hardware detection – Test to determine if a tool will attempt to implement design on faulty hardware.

Thorough literature research, surveys, and industry interviews, uncertainties and faults regarding the usage and/or operation of the tools has been compiled and analyzed. The case studies are focused on verifying the validity of these findings. The scope of these findings includes user interaction with the tool: if a user tries to implement something physically impossible, does the tool notify the user, alter the design to make it possible, attempt to implement the design, etc.? This leads to another topic in the scope of the case studies, does the tool have ¨awareness¨ of the hardware physical limits, or is this up to the user to oversee. For example, will the tool try to implement a component on a faulty piece of hardware? Will the tool

exceed the minimum transition time of the gate timing or account for a safety margin? These are just a few of the many identified concerns each of which traces to safety constraints or timing constraints.

The majority of the tools come in a package that encompasses everything from coding or formal requirements/design to redesigning through synthesis to testing the final implementation. The tools appear to be self contained, in the sense that they do their own verifications and testing, including even self validation of formal requirements/design. This raises the question: if the tool does all the design/redesign and it verifies its own design will it ever really know if it is correct? This also begs the question of the independence of the verification that is required by DO-178B and DO-254.

For example, the purpose of Real-Time Constraints case study is to determine if a circuit meets the timing constraints that the tool displays in the design report and that margin of safety is retained. The tools have the ability to specify the time that it will take a given operation to complete. If the bounds of the speed of the gates are pushed close to their minimums, the tool will redesign the circuit so that the delay is smaller. The subjects of interest are: where are these bounds, how close does the tool allow the design to get to the bounds, and is there a safety margin with actual delay and estimated delay?

The case study shall be implemented with an asynchronous ripple counter. An asynchronous ripple counter is a series of N flip-flops cascaded together so that the output of a flip-flop is clock input of the next one; this shall be done with five J-K flip-flops in toggle mode. The input to the ripple counter shall be an external pin that connected to a high frequency source. The final output of the counter shall be connected to an output pin. The expected output signal frequency shall be the input signal frequency divided by 2^N. During the initial implementation, a report is generated by the tool to establish the maximum input frequency and the signal path delay.

Since experiments are designed to run on variety of platforms (i.e. combination of tool/board), the instruction set shall be written so that it is platform independent. For each platform it will be necessary to create different configuration files to utilize the development board I/O's. The design shall be synthesized and configured to each platform independently. Each platform will require that timing constraints are set for the ripple counters. It is acceptable that each platform timing constraints will be different. A timing report will be generated by the tool.

This and all the following experiments require a radio-frequency (RF) generator, function generator, oscilloscope capable of analyzing RF or RF specific measuring equipment, and a logic analyzer. The implementation shall be tested in two iterations. The first iteration shall be done with the system default timings. The second iteration shall be done with the tightest allowable timing constraints. The tight timing constrains require the tool to alter the design so that higher frequencies are obtainable.

The experiment requires manual sweep of the input pin from 0 Hz to the highest obtainable frequency in the GHz range (higher than the maximum reported gate frequency of the hardware). During the sweep both the input and output wave of the component shall be scoped for accuracy and phase differences. The input frequency shall be increased until either the signal path time does not meet requirements or the output wave is not correct. The phase differences will yield the signal path time, the point of failure will determine the safety margins applied. All data and settings shall be recorded accurately for analysis.

5. Conclusion

If a system designer chooses to implement a design utilizing processors that are embedded inside significant hardware, the resulting system becomes a problem for safety-critical applications. Neither RTCA guidance DO178B (software) nor DO-254 (hardware) succeed in addressing the problems that are unique to a mixed system. Despite the design and verification efforts applying the appropriate hardware or software design tools, there is a significant concern that some errors can potentially slip through. This paper analyzes the related issues and describes a method and procedures to evaluate a tool with respect to real-time constraints, power analysis, and I/O issues for the purpose of qualification. One part of the solution might be that an anonymous database of known issues be compiled and that this database be used as a collective knowledge base for the use of software tools in design of complex electronic hardware for safety-critical systems. The case studies reported in this paper may be used to begin the system designer's knowledge base.

Acknowledgements

This work has been done under a contract supported by the Federal Aviation Administration DTFACT-07-C-00010. Contribution of MSE graduate student Jason Firanski to creation of the test cases is appreciated.

References

[1] Vahid F., It's Time to Stop Calling Circuits "Hardware", *Computer*, Vol. 40, No. 9, pp. 106-108, September 2007.

[2] Kornecki A. and Zalewski J., Software Development Tool Qualification from the DO-178B Certification Perspective, *CrossTalk – The Journal of Defense Software Engineering*, Vol. 19, No. 4, pp. 19-22, 2006.

[3] Jacklin S.A. et al., Development of Advanced Verification and Validation Procedures and Tools for the Certification of Learning Systems in Aerospace Applications, *Proc. AIAA Infotech@Aerospace 2007 Conference*, Arlington, Virginia, Sept. 26-29, 2005, Paper No. AIAA 2005-6912.

[4] Kornecki A. and Zalewski J., Experimental Evaluation of Software Development Tools for Safety-Critical Real-Time Systems, *Innovations in Systems and Software Engineering – A NASA Journal*, Vol. 1, pp. 176-188, 2005.

Applying Fault Injection to Study the Effects of Intermittent Faults

L.J. Saiz, J. Gracia, J.C. Baraza, D. Gil, P.J. Gil
Grupo de Sistemas Tolerantes a Fallos (GSTF) – Instituto de Aplicaciones de las Tecnologías de la Información y de las Comunicaciones Avanzadas (ITACA)
Universidad Politécnica de Valencia, Spain
e-mail: {ljsaiz, jgracia, jcbaraza, dgil, pgil}@disca.upv.es

Abstract

As new submicron technologies have allowed increasing processors performance and decreasing their size, their reliability has decreased. In this way, intermittent faults are expected to be an important challenge in modern VLSI circuits. Currently, intermittent faults due to manufacturing residuals and process variations have grown, being necessary to study their effects. Using as background faults observed in real computer systems, we have injected intermittent faults in the VHDL model of a microcontroller. The controllability and flexibility of VHDL-based fault injection technique permits to carry out a detailed analysis of the influence of some parameters of intermittent faults.

1. Introduction

The advances in integration techniques have allowed rising microprocessors operating frequency as well as reducing their size and power voltage, achieving in this way a greater performance. Nevertheless, these advances have had a negative impact on reliability. As the new submicron technologies shrink device sizes, the rate of occurrence of faults increases. Whereas the effects of permanent and transient faults have been studied in depth [1], the influence of intermittent faults is not considered very often. It is foreseen that intermittent faults (together with transient faults) will have a great impact in deep submicron technologies [2]. The complexity of the manufacturing process (that provokes residuals and process variations) and some wear out mechanisms can be the origin of such faults. Although errors induced by transient and intermittent faults manifest in a similar way, the last ones are activated repeatedly in the same place, and so they are usually grouped in bursts [3].

Our objective is to study the impact of intermittent faults, and compare their consequences to those provoked by permanent and transient faults. We have used VHDL-based fault injection due to its flexibility and the high observability and controllability of all the modelled components [4]. In addition, fault injection allows performing a systematic and exhaustive analysis of the influence of the parameters of intermittent faults on the system's behaviour.

This work is organised as follows. Section 2 describes the intermittent fault models injected. Section 3 explains the fault injection experiments, and Section 4 discusses the results obtained. Finally, Section 5 provides some conclusions and a proposal of future work.

2. Fault models

An important issue is the definition of the fault models to be injected, as they must be representative of real faults. This requires studying the fault mechanisms considering the features of the technology used. Nevertheless, and as far as we know, the causes and mechanisms of intermittent faults in new submicron technologies have not been studied so much. Intermittent faults can be related to wear out mechanisms that eventually end up in permanent faults [5]. But it is also becoming more and more necessary to keep in mind the influence of manufacturing defects, like process variations and residuals.

In this paper we have selected a set of intermittent faults frequently observed in real computer systems by means of fault logging [3]. Then, we have extrapolated their effects to fault models applicable to the logic/RT abstraction levels. In this way, we can inject and simulate these faults into VHDL models. Table 1 summarises these intermittent faults and their mechanisms and models.

Table 1. Intermittent faults mechanisms and models

Causes	Targets	Fault mechanisms	Type of fault	Fault Models
Residuals in cells	Memory and registers	Intermittent contacts	Manufacturing defect	Intermittent stuck-at
Solder joints	Buses	Intermittent contacts	Manufacturing defect	Intermittent pulses, short, open
Electromigration Barrier layer delamination	Buses I/O connections	Variation of metal resistance	Wear out-timing	Delay, intermittent short, open
Crosstalk	Buses I/O connections	Electromagnetic interference	Timing	Delay
Soft breakdown	Power supply voltage	Leakage current fluctuation	Wear out-timing	Indetermination, delay

3. Fault injection experiments

As stated above, intermittent faults manifest in bursts. So, to inject this type of faults, the following parameters must be configured (see Figure 1): the number of activations of the fault in the burst (we will call it *burst length*, or L_{Burst}), the duration of each activation (we will refer to it as *activity time*, or t_A), and the separation between two consecutive activations (we will name it as *inactivity time*, or t_I).

The different fault injection experiments have been carried out in the VHDL model of the 8051 microcontroller running the Bubblesort sorting algorithm as workload. Fault injection experiments have been carried out using a fault injection tool called VFIT (VHDL-based Fault Injection Tool) [6], that runs on PC computers (or compatible) under Windows®.

In order to study the effects of intermittent faults, we have calculated in all cases the percentages of *failures* provoked and *latent errors* (that is, errors that propagate to registers and memory but do not cause failures).

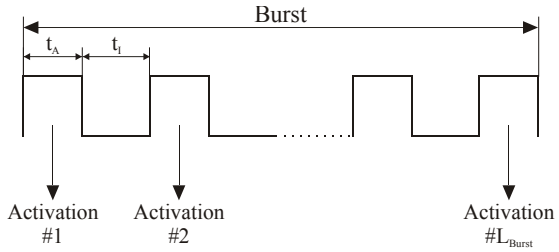


Figure 1. Description of the main elements of a burst

4. Results

Table 2 to Table 5 show the effect of intermittent faults injected in two types of targets: memory and registers (Table 2 and Table 3), and buses (Table 4 and Table 5). The fault models injected have been (see two first rows in Table 1): intermittent stuck-at (in memory and registers) and intermittent pulses (in buses). Time activity (t_A) has been generated randomly in three time intervals depending on the system clock cycle (T). Finally, single and multiple (in space domain, i.e. simultaneous faults in different targets) intermittent faults have been injected

Table 2. Results for single intermittent faults injected in memory and registers

Activity time (t_A)	[0.01T, 0.1T]	[0.1T, 1.0T]	[1.T, 10.0T]
Failures	0.1 %	0.4 %	3.9 %
Latent errors	41.9 %	8.9 %	0 %
Non effective	58.0 %	90.7 %	96.1 %

Table 3. Results for multiple intermittent faults injected in memory and registers

Activity time (t_A)	[0.01T, 0.1T]	[0.1T, 1.0T]	[1.T, 10.0T]
Failures	1.7 %	15.5 %	53.0 %
Latent errors	43.0 %	6.4 %	0 %
Non effective	55.3 %	78.1 %	47.0 %

Table 4. Results for single intermittent faults injected in buses

Activity time (t_A)	[0.01T, 0.1T]	[0.1T, 1.0T]	[1.T, 10.0T]
Failures	3.4%	28.2%	55.3%
Latent errors	2.5%	14.1%	16.3%
Non effective	94.1%	57.7%	28.4%

Table 5. Results for multiple intermittent faults injected in buses

Activity time (t_A)	[0.01T, 0.1T]	[0.1T, 1.0T]	[1.T, 10.0T]
Failures	13.4%	60.8%	89.4%
Latent errors	6.2%	19.8%	7.6%
Non effective	80.4%	19.4%	3%

The main observations that can be made are:

- As expected, the percentage of failures grows with the activity time, in both registers/memory and buses. We can also note that intermittent faults in buses have more impact that those in memory/registers.
- The percentage of latent errors does not show a clear trend with regard to the activity time. Whereas it decreases in registers/memory, it

rises slightly in buses. We think that this discrepancy is due to the workload.

- Multiple faults provoke a greater percentage of failures, which also was foreseeable. The difference is more pronounced in registers/memory than in buses.
- No significant differences in the percentage of latent errors are detected between single and multiple faults.

Finally, Figure 2 compares the impact of transient, intermittent and permanent faults injected in buses. In the case of transient faults, three different values of fault duration have been considered, using the same time intervals as for the activity time in intermittent faults. Some examples of transient and permanent fault models injected can be found in [6].

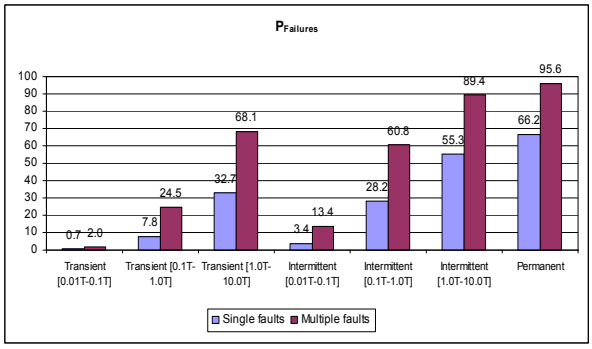


Figure 2. Comparison of the percentage of failures provoked by transient, intermittent and permanent faults in buses

From the figure, we can see that intermittent faults provoke a greater percentage of failures than transient faults. This is a logical result, as a burst of intermittent faults manifest like a sequence of transient faults in spite of having different origin. The effect of fault duration is also shown for all types of faults. Obviously, the greater impact is due to permanent faults because of their indefinite duration.

5. Conclusions and Future work

In this work, we have presented a case study of the effects of intermittent faults on the behaviour of a commercial microcontroller. The methodology used lies in VHDL-based fault injection technique, which allows a systematic and detailed analysis of the influence of different fault parameters. We have also compared the results to those obtained when injecting transient and permanent faults.

In the future, we have planned to extend this study in different aspects. First, we want to use other workloads in order to try to generalize the results. We also intend to analyse the effect of other parameters, like the burst length, the inactivity time, and the system clock frequency. Moreover, we think that it is important to inject other fault models, like delay, intermittent short and open. Finally, it is necessary to investigate more deeply the causes and mechanisms of intermittent faults related to new submicron technologies, in order to propose new fault models related to them.

6. Acknowledgement

This work has been partially funded by the Spanish Government under the project MCYT TEC2005-05119 “Estudio de las técnicas de inyección de fallos en modelos en VHDL: emulación de fallos en FPGA y uso de nuevas técnicas de simulación distribuida”.

7. References

[1] P.J. Gil et al., “Fault Representativeness”, *Deliverable ETIE2 of Dependability Benchmarking Project*, IST-2000-25245, 2002.

[2] C. Constantinescu, “Impact of Deep Submicron Technology on Dependability of VLSI Circuits”, in Proc. DSN 2002, pp. 205-209, Washington D.C., June 2002.

[3] C. Constantinescu, “Impact of Intermittent Faults on Nanocomputing Devices”, in WDSN-07, Edinburgh, UK, June 2007, http://www.laas.fr/WDSN07.

[4] A. Benso and P. Prinetto, eds., “*Fault Injection Techniques and Tools for VLSI reliability evaluation*”, Kluwer Academic Publishers, 2003.

[5] J.C. Smolens et al, “Detecting Emerging Wearout Faults”, 3rd Workshop on Silicon Errors in Logic - System Effects (SELSE3), April 2007.

[6] J.C. Baraza et al, “A Prototype of a VHDL-Based Fault Injection Tool: Description and Application”, *Journal of Systems Architecture*, vol. 47(10):847–867, 2002.

Deploying Automated Performance Analysis Techniques to Facilitate the Verification of Avionics Applications

Benjamin Gorry Nicola Herbert
BAE Systems Rapid Engineering, W374B, Warton Aerodrome, Warton, Lancashire, PR4 1AX
ben.gorry@baesystems.com, nicola.herbert@baesystems.com

Abstract

Assessing the performance of large-scale real-time systems can prove to be an arduous task. When the systems reside in a safety-critical domain, such as the field of avionics applications, they must perform to a high standard in order to satisfy correctness constraints. This paper proposes that the deployment of automated model-based performance analysis approaches may assist in assessing the performance of Avionics Applications.

1. Introduction

Acceptable levels of timeliness are imperative in modern day avionics applications. If a system, or part of a system, fails to perform as required within an allocated timeframe it may be labeled as faulty. In a real-time application it may be the case that the system is functionally incorrect in the real-time and un-timed domain or it may be the case that the system is functionally correct (in the un-timed domain) but does not perform to an acceptable degree of timeliness when real-time system constraints are taken into consideration. In addition to this, avionics applications are currently being developed in the form of Unmanned Air Vehicles (UAV) which make the use of complex probabilistic algorithms (such as Bayesian approaches). There will be a requirement to assess the correctness of these algorithms.

This paper discusses the possibility of deploying automated performance analysis approaches to assess the integrity levels of Integrated Modular Architectures (IMA) (with the idea being that we could deploy similar techniques to assess the performance of Advanced Avionics Architectures (AAvA) such as UAV). Firstly, we will discuss why automated model-based approaches to performance analysis could prove useful in the development of avionics applications. Following this we shall provide background on IMA approaches and the required Software Integrity Levels (SILs). We will then provide background on the approaches which are being considered. Finally, the direction that our work will take is listed.

2. Why Performance Analysis?

Current approaches to analysing the correctness of avionics applications involve the use of analysis techniques such as the SPARK Approach [1] and real-time testing [2]. To assess the run-time performance of avionics applications, testing approaches are commonly deployed. By adopting testing approaches it may not be practical or economically viable to test every possible combination of system parameters; the problem of a potentially unreliable system may therefore exist even after testing has been completed. Rushby [3] identified this as:

> *"The fallibility of testing becomes particularly acute in the case of distributed and real-time systems, such as the embedded systems used for spacecraft control"*

while Leveson [4] states that:

> *"testing for trustworthiness is an intractable task."*

Automated performance analysis approaches involve the development of models. By using a model-based approach to performance analysis three benefits are provided:

1. A model is more tractable than testing; it can be more easily adapted if the parameters of interest change.
2. Even if the performance analysis using the model does not provide satisfactory results, it

should allow us to 'focus' our testing on the area of interest.

3. A model is an abstraction of the system, therefore any damage done to the model will not affect the system. Testing however is carried out on the actual system; therefore it may be possible that a test case could damage the system.

The remainder of this section provides background on Integrated Modular Architecture (IMA) approaches to avionics systems development, the required Software Integrity Levels (SILs), and the analysis approaches which are being considered.

2.1. Integrated Modular Architectures

Traditionally, aircraft systems were developed such that individual hardware and software was designed to be compatible only with each other. The software was designed to run only on that hardware, so that it was

"tightly coupled to the underlying hardware" [5].

This is the traditional approach to aircraft system development. A different approach is to adopt Integrated Modular Architectures (IMA). This approach suggests using a network of standard hardware and software modules, each of which has well-defined interfaces [6]. Each of the software applications can be executed on any of the standard hardware modules, the main advantage being that applications can be reconfigured to any hardware module in the event that the hardware module on which a particular software module is executing fails. As all software is 'hardware transparent', this

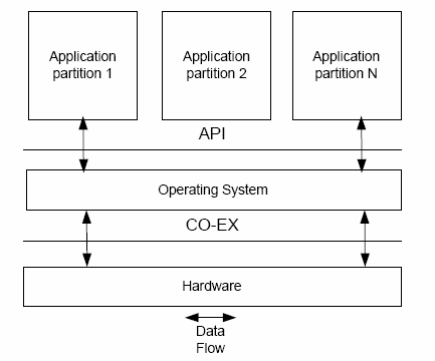


Figure 1: Example Integrated Modular Architecture [5]

approach makes it more cost effective than in the federated approach when the time comes to upgrade obsolete hardware devices. One suggested architecture for IMA is described in the civil avionics standard ARINC 653, as described in [5], as shown in Figure 1. As discussed in [1, 2], in this architecture, each hardware module has:

- A number of software applications, each of which is partitioned from other applications and the operating system.
- An Application Programming Interface which allows the software applications access to resources.
- An Operating System Layer, which provides a standard set of basic services such as process scheduling and communications.
- A Core Executive (CO-EX) which interfaces between the hardware and the operating system.

2.2. Software Integrity Levels

In avionics applications it may not be necessary to achieve the highest level of integrity for each section of a system. Software Integrity Levels (SILs) are defined in Defence Standard 00-56 Issue 2 [7] as:

"An indicator of the required level of safety integrity".

Where [8] describes integrity as:

"Integrity is the probability of a system satisfactorily performing the required critical functions. The integrity level therefore provides a qualitative indicator that the system meets its safety objectives."

SILs are used to provide a software developer with a guide as to the level of rigour that should be involved in developing software. SIL levels for a system are determined from an initial hazard and risk analysis of the software that is to be developed. This analyses the risks that the system will pose in the environment in which it is to be used, and is based on the severity of identified hazards, and the probability that the hazardous situation might arise. It therefore takes into account the seriousness of any accidents that might occur if the system fails, as well as the probability of an accident type occurring. The determination of SILs is illustrated in the Figure 2, taken from [7].

The SIL level, once identified, is used to dictate the software development and verification processes that should be followed during software development. The chosen process should ensure that the software is developed with a level of rigour commensurate with the level of risk. The chosen process involves the

application of appropriate design rules, methods, tools and techniques [8]. Suggested activities are prescribed

Failure Probability	Accident Severity			
	Catastrophic	Critical	Marginal	Negligible
Frequent	SIL 4			
Probable		Level SIL 3		
Occasional			Level SIL 2	
Remote				
Improbable			SIL 1	

Figure 2: Safety Integrity Levels [7]

in some standards. For example, according to [7], for SIL 4 the software requirements and designs should be formally specified, whereas for SIL 3, they only need to be semi-formally specified.

2.3. Performance Analysis Tools

By using automated approaches to model analysis we can eliminate some of the shortfalls of a traditional testing approach. We can specify verification properties which must hold for all possible combinations of events in a model; these will eliminate the need to test each individual combination of events.

Traditional automated approaches to verifying models of systems have included model checking tools such as SPIN (Simple Promela Interpreter) [9]. Tools such as SPIN have proven useful when analyzing systems for correctness properties such as safety and liveness. One downfall to such an approach is that if a property fails to hold a counter-example is simply presented. By using performance analysis tools, a more informative answer such as *"what is the probability of event x occurring?"* can be obtained via the specification of correctness properties. This will provide a probabilistic estimation rather than a basic yes/no answer to a correctness property.

The three performance analysis tools which are under consideration are the PEPA (Performance Evaluation Process Algebra) Workbench [10], PRISM (PRobabalistIc Symbolic Model checker) [11], and SPNP (Stochastic Petri Net Package) [12]. These tools offer differing approaches to model development and analysis while utilizing the same underlying probabilistic modeling structure, Continuous Time Markov Chains (CTMCs) [13]. A CTMC is represented as:

$$<S, S_0, R>$$

Where S is a finite set of states in a model, S_0 is the initial state in a model, and R is the transition rate matrix (each element of the transition rate matrix assigns a rate (time) of making a transition from one state in the model to the next). We shall now discuss how PEPA, PRISM, and SPNP make use of CTMCs.

In PEPA, each model defines a labeled multi-transition system which can be read as a CTMC. A model is used to describe a system as an interaction of components which engage in activities. Components and activities correspond to states and transitions in the underlying CTMC. This CTMC can be solved for a set of defined performance measures, common choices of which are throughput and resource utilization. In PEPA, expressions can be constructed which describe the behaviour of components. The PEPA Workbench [10] provides support for the reasoning of PEPA models.

In PRISM, models consist of a series of modules which contain local variables. These local variables determine the state of a module at any given time and the global state of a model is determined by the local state of all modules. The behaviour of a module is specified by a set of guarded commands. Currently, PRISM models are produced in the form of CTMCs. To analyse a PRISM model we specify correctness properties by using either the temporal logic PCTL (Probabalistic Computation Tree Logic) or CSL (Continuous Stochastic Logic) [13]. In CSL the operators P and S can be used. P represents the probability that a certain type of behaviour may occur in the model, it allows us to reason about different paths through the model. S is used to reason about the long-term behaviour of a model.

The SPNP can be used to analyse the performance of CSPL (C-based Stochastic Petri net Language) models. SPNP supports the specification and solving of SRN (Stochastic Reward Net) models where SRNs are a class of Petri nets where the model definition includes both the structural and timing behaviour of a system. In SPNP a model can be formulated by using parameters. The model can then be solved numerous times by substituting different values for each parameter or a combination of parameters. Two main model analysis techniques are offered by SPNP, discrete event simulation and analytic-numeric simulation. Discrete event simulation allows us to study the behaviour of a system in discrete steps (up until a point in time). Analytic-numeric solution allows us to specify a SRN as a CTMC, this eliminates the need for analysis in discrete steps.

It is hoped that by deploying the capabilities of these tools when analysing IMA (and eventually AAvA), we can determine if sections of an avionics

application (or the application as a whole) conforms to its desired SILs. Since the SIL for different sections of an application may differ, performance analysis could allow us to assess these individual sections to determine if they conform to their desired SIL. Also, since each of the performance analysis approaches which have been discussed in this section are automated, allowing us to analyse every combination of model parameters, the use of these approaches could possible identify system resources which are being under-utilised. These resources could then be better deployed, making the application more efficient.

3. Direction of Work

The immediate plan of work involves four stages. Firstly, the subsets of the system source code languages which are most commonly used will be identified. Following this, a number of mapping relationships between the constructs of these languages and the PEPA, PRISM, and SPNP model representations will be defined. The third stage of this work will be to decide how to automate the extraction of models for performance analysis; during this stage we hope to build on the work discussed in [14]. Finally, a number of case studies will be analysed to assess precisely where we can deploy automated approaches to performance analysis. Because of the modular nature of many avionics applications, it is envisaged that our proposed approach will be viable for assessing the performance of individual modules or sections of a system as well as the system as a whole. We aim to make use of case studies which have previously identified performance issues to assess if we can discover the issues by using our approaches.

The work described in this paper is not envisioned to be a panacea to the problem of system verification. It is hoped that the efficient deployment of automated approaches to systems performance analysis will be a complimentary approach to other techniques such as [1] and real-time testing approaches. If we could assess the performance levels of a section of an avionics application we could then compare this against the required SIL and make an informed choice on how to further assess the application.

4. Acknowledgements

The authors wish to acknowledge the interesting discussions that have taken place with various members of the Systems and Software Safety Group (SSSG) and Rapid Engineering Capabilities (REC) Group within BAE Systems.

5. References

[1] J. Barnes, High Integrity Software: *The SPARK Approach to Safety and Security*, Addison Wesley, 2003.

[2] H. Kopetz, *Real-Time Systems, Design Principles for Distributed Embedded Applications*, Kluwer Academic Publishers, 1997.

[3] J. Rushby, *"Formal methods for Dependable Real-Time Systems"*, International Symposium on Real-Time Embedded Processing for Space Applications, pg 1-16, 1992.

[4] N. Leveson, Safeware – *System Safety and Computers*, Addison-Wesley Publishing Company Inc., 1995.

[5] J. Blow A. Cox P. Liddell, *Modular Certification of Integrated Modular Systems*, Constituents of Modern System-safety Thinking, Springer London, 2007.

[6] P. Conmy. J. A. McDermid., High Level Failure Analysis for Integrated Modular Avionics, Sixth Australian Workshop on Industrial Experience with Safety Critical Systems and Software, 2001.

[7] Defence Standard 00-56 Issue 2, *Safety Management Requirements For Defence Systems*, 1996.

[8] Y. Papadopoulos J. A. McDermid, *The Potential For A Generic Approach To Certification Of Safety-Critical Systems In The Transportation Sector*, Journal of Reliability Engineering and System Safety, Volume 63, Number 1, 1999.

[9] G. J. Holzmann, *The SPIN Model Checker, Primer and Reference Manual*, Addison-Wesley, 2003.

[10] S. Gilmore J. Hilston., *The PEPA Workbench: A Tool to support a Process Algebra-based Approach to Performance Modelling*, 7th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation, pg 353-368, 1994.

[11] M. Kwaitkowska G. Norman D. Parker, *PRISM: Probabalistic Symbolic Model Checker*, Proceedings of TOOLS, pg 200-204, 2002.

[12] G. Ciardo J. Muppala K. Trivedi, *SPNP: Stochastic Petri Net Package*, Third International Workshop On Petri Nets and Performance Models, pg 142-151, 1989.

[13] J. J. M. Rutten M. Kwiatkowska G. Norman D. Parker, *Mathematical Techniques for Analyzing Concurrent and Probabilistic Systems*, American Mathematical Society, 2004.

[14] B. Gorry A. Ireland P. King, *PARTES : Performance Analysis of Real-Time Embedded Systems*, Proceedings of the 4th International Conference on the Quantitative Evaluation of SysTems (QEST), pg 271-272, 2007.

Integrated Modular Processing for High Performance, High Integrity Control (IMPPIC)

Peter Hubbard
Loughborough University
P.D.Hubbard@lboro.ac.uk

Matthew Mapleston
BAE Systems plc
M.Mapleston@lboro.ac.uk

Roger Goodall
Loughborough University
R.M.Goodall@lboro.ac.uk

Roger Dixon
Loughborough University
R.Dixon@lboro.ac.uk

Abstract

The next generation of Avionic systems promises to be more complex and interoperation more common and demanding than the last. As such, key drivers within design of these futuristic systems are to reduce lifecycle costs and improve mission and operation performance. Integrated Modular Systems (IMS) present a solution to the above problems, but the knowledge base requires further maturity before its full benefits can be realised. This paper details ongoing research directed at the problem of managing fault tolerance mechanisms within IMS. The research is being performed as part of a project known as IMPPIC, which has been undertaken as part of a PhD and funded in collaboration by BAE Systems and the Engineering and Physics Research Council (EPSRC). The aims of this project are to provide a tangible demonstration of Fault tolerance within an IMS that has the demanding task of maintaining controlled magnetic suspension of a Maglev rig..

1. Introduction

Modern avionic systems have been progressively migrating from the well understood and well used federated systems to an integrated modular solution for at least the past decade. The drivers for this next generation of military avionics are to reduce lifecycle cost, improve mission performance and improve operational performance.

The next generations of civil and military avionics generally feature Integrated Modular System (IMS) architecture; however the military domain is more challenging than that enjoyed by civil avionics despite their apparent synergies. Military avionics systems operate in harsher physical environments e.g. high g, vibration, temperature extremes etc. and must consider enemy threats, battle-damage and the safe operation of dangerous weapons. Military avionics must also be designed under tighter engineering constraints e.g. available volume, power, weight etc. The operating context and nature of the systems themselves combine into a highly challenging engineering problem even before factoring in the drive to continuously decrease costs throughout the lifecycle and improve operational and mission performance.

One of the areas to be clarified with regards to the implementation of IMS is the management of faults throughout the system [1]. Three functional areas have been identified that must be addressed to achieve fault management in IMS, namely Fault Avoidance and Removal, Fault Tolerance and Fault Treatment [2].

The IMPPIC project has been commissioned to address these issues as part of a PhD that is jointly funded by BAE Systems and the Engineering and Physical Sciences Research Council (EPSRC) of the UK Government, and is situated in the Systems Engineering Innovation Centre (SEIC), a partnership facility between Loughborough University, BAE Systems and the East Midlands Development Agency. The goals of IMPPIC are:

- Implement a COTS based IMS development facility for a Maglev (Magnetic Levitation) system.
- Develop fault management solution within IMS
- Provide a tangible demonstration on a test rig.

This paper will introduce the concept of IMS in section 2 and then discuss the target application, namely the Maglev rig, in section 3. The IMS implementation used within this project will be outlined in section 4 and finally the fault management activities that are in design as part of the IMS will be presented in section 5.

2. Integrated Modular Systems

Systems based upon an IMS architecture feature standard (form, fit, function and interface) hardware modules connected to each other and various sensors and actuators by flexible communication networks. A layered software architecture, known as the three layer stack, and demonstrated in Figure 1, instantiated on each hardware module, provides standard services to application software independently of the underlying hardware. Application software processes are instantiated on physical resources and configured, according to a system blueprint, under the control of distributed System Management functionality. System management, together with the inherent flexibility of systems using an IMS architecture, enables automatic system reconfiguration in response to events in order to optimize system functionality, resource usage and dependability under different circumstances.

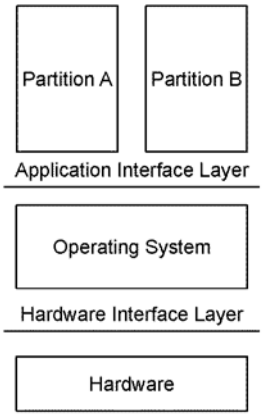


Figure 1 - Three Layer Stack

3. Maglev Application

The Maglev rig was chosen as a target application for the IMS as it will be a challenging real time control task and requires multi-input multi-output, deterministic execution in order to maintain magnetic suspension. This application has inherently less risk and complexity than the control of a platform such as an modern aircraft, whilst still providing a demanding task for purposes of demonstration. In addition, the Maglev system is open-loop unstable, as with some modern military aircraft, meaning that any failure of the controller will result in immediate failure. One of the principles of IMS is that it should be non-platform specific [3] so the systems management concepts demonstrated within this project will be comparable to scenarios within a full avionics system.

The IMS will be given the task of controlling the magnetic suspension of a 200kg magnetic levitation (Maglev) rig. Magnetic suspension is facilitated by 4 dual wound magnets, each magnet with airgap sensor, dual flux search-coil sensors, dual temperature sensors and dual hall effect current sensors.

The control structure of the rig has been derived from mathematical models and will consist of a 66Hz bandwidth flux loop, nested within a 13Hz bandwidth airgap loop for each magnet. There is also the potential to extend this full vehicle control structure to processing that introduces an active suspension component to the rig [4].

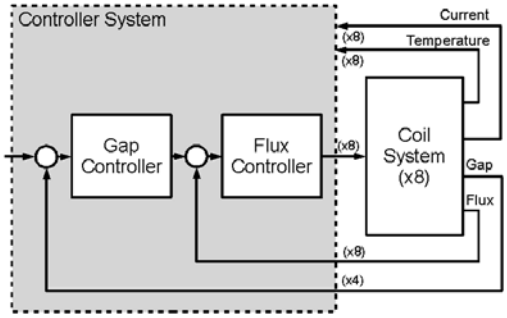


Figure 2 - Simplified Maglev Controller Schematic

4. IMS Implementation

Implementation of a full specification IMS as a development facility was beyond the scope and time frame of the project. As such, a representative IMS solution has been implemented with COTS that contains the key functional components required for testing, and will meet the demands of the Maglev controller.

The IMS has been developed based around a standard PC motherboard using a LabVIEW real time solution as the operating system. This format allows the easy integration of I/O in the form of National Instruments Data Acquisition (DAQ) cards and provides the necessary drivers to interface with communications systems such as Ethernet.

The LabVIEW real time operating system (RTOS) is not designed as an IMS middleware solution, and as such, some custom code generated in LabVIEW, known as a Virtual Instrument (vi) will provide the IMS systems management facility. Figure 3 shows how the configuration described matches the classic three layer stack.

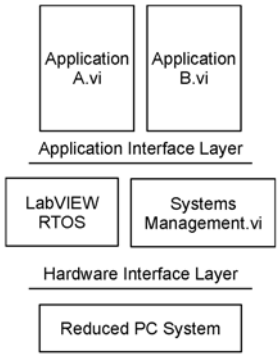


Figure 3 - Implementation of IMS

By replicating this modular set up across a network, a small scale IMS is created, as shown in Figure 4.

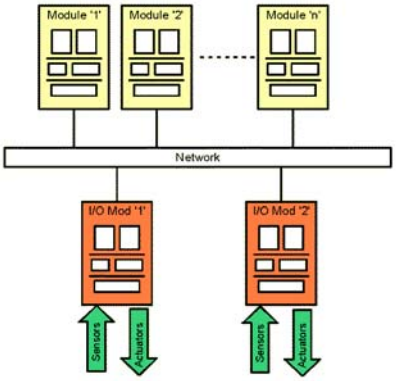


Figure 4 - IMS Overview

5. Fault Management

Three areas have been addressed as required components to achieve fault management in IMS [2]: Fault avoidance and removal, Fault tolerance and fault treatment. The area focused on in this project is the issue of fault tolerance.

The previous generation of avionics, based upon federated architectures, generally achieved fault tolerance through built-in test functions and static redundancy of critical functions. Avionics systems based upon IMS architecture concepts can be equipped with novel responses to fault conditions during operation that utilize the modularity and reconfiguration capabilities while achieving traditional dependability requirements. For example, dynamic redundancy can be used to isolate faulty system components and reconfigure the system to provide reversionary functions or reintroduce critical functions at the expense of other less critical functions. Additionally, online diagnostic functions can be applied to isolated components not contributing to required function, thereby avoiding unnecessary diversion of resources during normal operation. Graceful degradation of the system under fault conditions can be used to maximize mission performance and contribute to improved operational performance by extending the Maintenance Free Operating Period (MFOP).

A major drawback is that error propagation becomes more severe the more integrated a system becomes, and methods of partitioning areas of the system, e.g. atomic actions, are required.

Fault tolerance at runtime will be performed via three stages: detecting and isolating failure, reporting error and tolerating failure.

5.1 Detection and Isolation

For the purposes of the experiment, the system is monitoring for a small, finite number of discrete failure modes (i.e. the component will be completely failed or completely working). It is intended that during the demonstration of the project, these errors will be injected into the system, and the system reaction observed.

There is information within the literature about fault detection, and how to identify source problems for a variety of symptoms. As the focus of this project is on the reaction methodology to detected faults, the assumption will be made that only a small number of failure modes exist, and the problems of complex interaction of failures will largely be ignored.

Isolating the failure to prevent the propagation throughout the system is a serious issue in IMS. In this project, isolation is performed via two main methods; monitoring the execution of the applications and checking all data against redundant processing channels for validation. As many functional components can operate a single processing module, there is a potential for an application crash to effect other functions. In a true IMS, partitioning is created between applications such that this does not occur.

5.2 Error Reporting

Decision making, in particular the decision on how to react to error signals, will be performed at a central location by a full authority health manager. In this project, the health management system is looking for a finite number of identifiable trigger events in order to activate a reconfiguration process to tolerate the faults, and maintain a high level of redundancy.

All error reports will be sent to this central location and all configuration commands will generated here. In addition, an activity log will be kept such that the

behaviour of the system can be analysed at a later stage.

5.3 Fault Tolerance

The main method of fault tolerance to be implemented is system reconfiguration. This does not remove the fault, but will mask the symptoms in order to maintain safe, reliable service until the fault can be removed. This process demonstrates a practical use of configuration to improve on traditional approaches to redundancy.

There are two aspects to this problem. Firstly, the system needs to be able to identify symptoms correctly such that the subsequent reconfiguration removes the source fault. Secondly, the system needs to reconfigure dynamically at run time whilst causing no disruption to service (i.e. loss of air gap).

One method of managing this reconfiguration is to prepare a number of configurations that can be reverted to immediately should a trigger event occur [5]. Although there are still a number of safety issues to be considered with this approach, this will be the method implemented initially. It is intended that the development of the system progresses such that the systems manager can generate at run-time, the next best configuration based on the current configuration and likely failures.

The major flaw in this development is the capacity to recognise all combination of error reports as a particular systems component failure. At this stage, it is only possible for the systems designer to prepare for a finite number of faults, and there is a potential for a complex system interaction to yield an unexpected series of trigger events that may confuse the system. Research is being performed in this area to reduce this knowledge gap.

The decision-making for systems reconfiguration is made by a central systems management component, as described previously, presenting a further complication of a single point failure scenario. It is therefore necessary to not only consider the systems management of the IMS applications, but the management of the systems management function itself. (“quis custodiet ipsos custodes”? - who watches the watchmen?).

A proposed solution to this is to have a master/shadow systems management configuration across the distributed network. The shadow systems managers will simultaneously receive the same information as the master systems manager in order to mimic and check its output to ensure it is issuing commands that fit their analysis. Should the Master fail, or be deemed to be failed by consensus of Shadow managers, one of the other modules will be able to assume the Master status immediately.

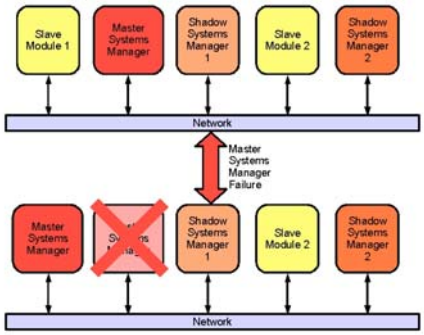


Figure 5 - Systems Management Reconfiguration

Figure 5 shows the benefit of reconfiguration to maintain redundant channels in the presence of faults. A slave module, a module in this case which performs minimal and mainly localised systems management activities, can be upgraded in functionality to perform the shadow master functionality and restore the redundant channel.

5. Progressions

The design detailed above is work in progress and is yet to be implemented in full. The work at the moment is focusing on the integration of designed systems elements to form the overall deliverable and is expected to be completed later this year.

References

[1] D. M. Nicholson, "Health Monitoring for Reconfigurable Integrated Control Systems," 2005.

[2] M. Mapleston, "Fault management in integrated modular systems," in *Sixth Engineering Dependable Computing Conference*, 2006, pp. 79-80.

[3] D. Field, "Integrated Modular Avionics A System Integrator's Perspective," 1998.

[4] R. M. Goodall, "Dynamics and Control Requirements for EMS Maglev Suspensions," 2004.

[5] D. M. Nicholson, P. Hollow and J. McDermid, "Approaches to Certification of Reconfigurable IMA Systems," 2000.

Treatment of Dependability and Uncertainty in Probabilistic Safety Assessment of Delayed System

Robertas Alzbutas
Kauno Technologijos Universitetas, 50 Studentu, LT- 51368, Kaunas, Lithuania,
Lietuvos Energetikos Institutas, 3 Breslaujos, LT-44403, Kaunas, Lithuania, robertas@mail.lei.lt

Vytautas Janilionis, Jonas Rimas
Kauno Technologijos Universitetas, 50 Studentu, Lt- 51368, Kaunas, Lithuania,
Vytautas.Janilionis@fmf.ktu.lt, Jonas.Rimas@fmf.ktu.lt

Abstract

In order to study the analytical modeling and the simulation of hybrid (continuous-discrete) systems as well as for dynamic reliability considerations, recently the Theory of Stimulated Dynamics (TSD) or so called Stimulus-Driven Theory of Probabilistic Dynamics (SDTPD) has been developed. The theory deals with the concept of stimulus and delays and how it can be implemented. A formerly used Markovian approach is provided in order to adapt TSD to practical applications, mostly in the context of dependability and uncertainty treatment in the Probabilistic Safety Assessment (PSA). Since the application of TSD to the PSA concepts needs a formal approach, definitions and additional investigation related to dependability and uncertainty treatment are further considered. The concept of sequence of dynamics is introduced as a possible way to present TSD-based dependable and uncertain delayed system in direct relation to PSA. The considered sequence and scenario development should efficiently use the available computer resources and at the same time allow collect information for scenario analysis considering uncertainty and dependability.

1. Introduction

1.1. PSA issues and Markov process

In the framework of Probabilistic Safety Assessment (PSA) used for Nuclear Power Plants (NPPs) the event tree technique has been developed mainly to represent and investigate different scenarios, which lead either to the accident, or to safe or intermediate states influenced by failures of safety/protection technical systems and human errors. Considering severe accidents, there is an objective to investigate the rarest scenarios of components and systems failures leading to almost improbable but very severe accidents. Due to the strong coupling existing along an accident scenario between stochastic (failure) events and the dynamics of physical processes, the traditional event tree methodology (and its associated tools) is not capable to cope with such a scenario assessment. Thus, extended PSA approaches are considered in order to cope with the related issues and to perform independent regulatory evaluations (Alzbutas et al. 2007 [1], Izquierdo et al. 2002 [7]).

As for the insight it brings into safety-related issues, the traditional PSA approach, in practice, reduces the calculation of safety objectives exceedance frequencies to the probabilistic quantification of initiating events and sequence headers related to basic events frequencies. The headers describe the status of essential safety features like the availability of safety barriers (functions) in terms of safeguard systems configurations, assuming they are demanded during the accident transients.

The most commonly used methods for PSA are based on the assumption, that the basic events are functionally independent of each other. This is the typical quantification approach that has proved very useful, particularly for very large systems with many basic events. However, basic events independence cannot always be demonstrated. For instance, for basic events

representing component failures, independence implies the failure rate λ of a component to be independent on the state of the other components. This assumption does not often hold, and Markov models are mainly used to account for time dependence of the reliability and unavailability functions. Using Markov processes, such an allowance can come out naturally and does not pose additional complexities.

In the case of systems with a failure, repair, test and maintenance cycle, it is possible to use an assumption that the transfers from state to state follow a Markov process. The equations to be solved may be described in their differential form by:

$$\begin{aligned} &\frac{d}{dt}\pi_{\vec{j}}(t) = -\lambda_{\vec{j}}\pi_{\vec{j}}(t) + \varphi_{\vec{j}}(t)\ ; \\ &\lambda_{\vec{j}} \equiv \sum_{\vec{k}\neq\vec{j}} p_{\vec{j}\to\vec{k}}\ ;\ \varphi_{\vec{j}}(t) \equiv \sum_{\vec{k}\neq\vec{j}} p_{\vec{k}\to\vec{j}}\pi_{\vec{k}}(t)\ ; \end{aligned} \qquad (1)$$

where $\vec{j}$ is the system state vector, composed of the set of component states, $\pi_{\vec{j}}(t)$ is the probability of the system being in state $\vec{j}$ at time t and $p_{\vec{j}\to\vec{k}}$ is the transition rate from state $\vec{j}$ to state $\vec{k}$. The term $\varphi_{\vec{j}}(t)$, as defined in Equation 1, is called the ingoing density, i.e., the instantaneous frequency at which state $\vec{j}$ is entered from any other state at time t.

In a Markov stochastic process, the probability for the system to stay in a given state during a given sojourn time is independent of the time at which the state is entered, so state probabilities are independent of the past history (memoryless stochastic systems).

The assumption of a Markov approach (i.e. the assumption that the probability that a system will transfer from one particular state to another depends only on the initial and final states of the transition) holds major simplification of the simultaneous equations describing the state space diagrams. The terms of the evolution equations for each state in this case depend only on the state itself, on the possible states immediately preceding and following the one under consideration, and on the rates of transfer between these states.

However, even these simpler equations may not be soluble in closed form, e.g. if the rates of transfer between states are time-dependent functions. Solutions would have to be found by iterative, numerical methods and the solution for each state would be a function describing the variation with time of the probability of that state.

1.2. Extensions of the Markov approach

A number of different methodologies were proposed in order to deal with events and the time that elapses during the evolution of dynamics. Most known theoretical background of these methodologies to treat and analyze dynamic systems was based on the Markovian approach. For instance, the Theory of Probabilistic Dynamics (TPD) (Devooght & Smidts 1996 [5], Izquierdo et al. 1996 [7]) was extensively investigated in order to perform analytical modeling related to the analysis of the systems reliability and safety. Dynamic reliability techniques (Devooght 1997 [4], Labeau et al. 2000 [10]) have been developed in order to study the reliability parameters of complex dynamic systems having continuous processes and discrete failure events interacting with each other. In dynamic reliability theory, the concept of reliability includes the interaction existing between the sequence of dynamics and events, such as the crossing of the border of a safety domain in the space of the physical variables, as well as the transitions between dynamics. The large number of states and consequently of transition probabilities that are to be evaluated is the most important limitation of using Markov techniques for large sets of system components. Thus, grouping of dynamics becomes necessary and some general grouping techniques have been further devised and discussed.

Extensions of Markov processes have been developed for cases where the transition rates depend on process variables, i.e. when the TPD is valid. It has been shown that the typical event tree approach (i.e., setting the probability of the demand to 1 or 0) remains valid for sequences of events instantaneously triggered by setpoint crossing, with the assumption that a setpoint is deactivated immediately after its associated event has been triggered (Labeau & Izquierdo 2005 [9]).

However, in some unfortunately frequent cases, there is a stochastic time delay since the activation of setpoints before the actual occurrence of the corresponding events and/or they are not deactivated after the events. For instance, operators introduce delays in taking potentially different actions after alarms that may remain concurrently activated. In these cases, stochastic delays complicate the situation.

More generally, the same situation occurs whenever for the events to occur, some conditions ought to be fulfilled that depend on the accident transients followed.

These conditions may persist after the events. A typical example is the occurrence of combustion phenomena only if flammability conditions are met, with delays potentially resulting from stochastic ignition conditions and with potential for multiple combustions if the flammability conditions persist. Those more general conditions (including setpoints as particular cases) may be considered as stimuli for the events. When accident transients reach those conditions, we speak of the *stimuli activation*.

Because stimuli activation conditions the events, the history of stimuli activations during the accident transients do matter in calculating the frequencies and extensions of the Markov process equations accounting for these features are necessary. Those extensions constitute the so-called Stimulus Driven Theory of Probabilistic Dynamics (SDTPD) (Labeau & Izquierdo 2005 [9]).

Indeed, SDTPD provides mathematical balances to calculate the probabilities per unit time of entering states with specified activated stimuli and correlate them with each other. In addition, in the calculation of the exceedance frequencies, the probability of the demand and the fraction of damage transients are not factorized but rather embedded in the activation of the stimuli. Exceeding safety objectives is a particular case of stimulus, so SDTPD has the potential for analyzing multiple safety objectives. Work is in progress to better relate the theory with the PSA itself.

More recently the simplification of SDTPD or Theory of Stimulated Dynamics (TSD) was developed for the analytical modeling and the simulation of hybrid (continuous-discrete) systems (Alzbutas & Janilionis 2007 [2], Alzbutas & Janilionis 2006 [3]). The theory at first deals with instantaneous and random variations of process variables; then, it introduces the concept of stimulus and how it can be implemented. Both a semi-Markov and a non-Markovian treatment are provided in order to adapt TPD for practical applications, mostly in the context of PSA (Labeau & Izquierdo 2005 [9]). The development of TSD as well as related methods and simulation methodologies has been agreed as a basis in the perspective of its applications for PSA and severe accident analysis.

Since the application of TSD-related methods to the traditional PSA concept (Izquierdo & Labeau 2004 [6]) needs a formal approach, the new definitions are specified and additional investigation related to uncertain scenario and sequence grouping is performed. The sequence of dynamics is introduced as a possible way for TSD based results presentation. The issues of PSA and its relation to TSD results are discussed.

2. Stimulated and delayed system

At first, let us determine analytically the basic ingredients of stimulated dynamics and consider them in relation to each other. Events are associated to an instantaneous change of the dynamics due to stimulus. System dynamics is determined by the law of process variables evolution, which can be indexed by an integer $i \in N$. Process variables $\overline{x}$ can be governed by a set of deterministic equations, e.g. $d\overline{x}/dt = f_i(\overline{x})$, $\overline{x}(0) = \overline{x}_0$, $\overline{x} \in R^N$. Event e is defined as transition of dynamics, in particular case from dynamics i to dynamics j at time t, i.e. $e: C \to C$. Random event is the event whose occurrence is related to complex nature, which is modeled stochastically. An example of random event is a time distributed failure occurrence. Deterministic event is induced by the deterministic rules. An example of such an event can be related to time moment when a threshold pressure or temperature is reached and safety functions is activated.

Paths of dynamics or evolution of process variables between transitions are associated to deterministic transients. Paths depend on initial and boundary conditions, which are associated to the initial states of process variables. Transitions are associated to significant changes in dynamic status, i.e. the end of one dynamics and the beginning of next one. Sequence of transitions is related to the sequence of events, part of which can be introduced artificially just for simulation purposes.

To describe transition (in particular case due to event e) from dynamics i to dynamics j at time t, let us define the transition rate between dynamics $j \to i$ as $p(j \to i \mid \overline{x})$, which characterizes the part of the sequence of transitions (or the sequence of events if all transitions are related to events) corresponding to the transition between two dynamics $j \to i$. Then the total transition rate out of dynamics j is equal to the sum of all transition rates out of dynamics j to any $i \neq j$:

$$\lambda_j(\overline{x}) = \sum_{i \neq j} p(j \to i; \overline{x}) \tag{2}$$

All the information on the any dynamics i is accumulated into the probability density function $\pi(\overline{x}, i, t)$

of the dynamics i at time t and with process variables in $d\bar{x}$ about $\bar{x}$.

Before description of sequence of events, two transition densities related to any dynamics i can be also introduced:

– Outgoing density out of dynamics i to any $k \neq i$ at $(\bar{x},t)$:

$$\psi(\bar{x},i,t) \equiv \sum_{k\neq i} p(i \to k \mid \bar{x})\pi(\bar{x},i,t) = \lambda_i(\bar{x})\pi(\bar{x},i,t). \quad (3)$$

– Ingoing density into dynamics i from any $\mathrm{j} \neq \mathrm{i}$ at $(\bar{x},t)$:

$$\varphi(\bar{x},i,t) \equiv \sum_{j\neq i} p(j \to i \mid \bar{x})\pi(\bar{x},j,t) = \sum_{j\neq i} \hat{p}(j \to i \mid \bar{x})\psi(\bar{x},j,t). \quad (4)$$

To describe event in relation to stimulus, there is need to note that stimulus covers any situation that potentially causes, after a given time delay, an event to occur (see Fig. 1). In the usual formulations of the theory of probabilistic dynamics the change in the dynamic behavior of the physical process variables occurs with no delay after the solicitation causing a branching in the continuous event tree. The main concept introduced in stimulus driven theory of probabilistic dynamics is that of stimulus, which must take place prior to the actual transition between two dynamics, i.e. system configurations corresponding to different dynamic evolutions.

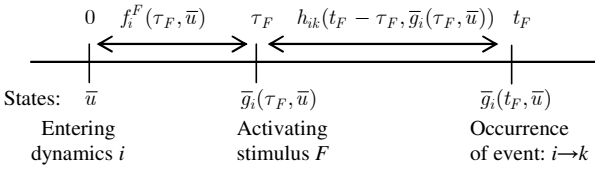


Figure 1. Stimulus F activation and delay before event.

Let Φ be the set of all stimuli to be accounted for in the process evolution following the occurrence of a given event related to the transitions between dynamics. Denote by $f_i^F(\tau_F;\bar{u})$ the probability density function of activating particular stimulus $F \in \Phi$ at states $\bar{g}_i(\tau_F,\bar{u})$ after time τ_F spent in dynamics i which was entered at state $\bar{u}$. Also define $h_{ik}^F(t_d;\bar{u})$ - probability per unit time of having a time delay $t_{dk} = t_F - \tau_F$ between stimulus F activation at time moment τ_F and occurrence of event, i.e. transition of dynamics $i \to k$, if stimulus F was activated at states $\bar{u}$.

$$h_i^F(t_d;\bar{u}) = \sum_{k\neq i} h_{ik}^F(t_{dk};\bar{u}) \quad (5)$$

where $h_i^F(t_d;\bar{u})$ is probability density function of the delay t_d between stimulus F activation at time moment τ_F and in the same conditions leaving dynamics i at time moment t_F.

Sequence of events consists of events, with system dynamics in between. Every sequence of events should be unique, but can lead to the same consequence. Sequence of event is an instance of the system status evolution through time line, i.e. $E = e \times T$. Space of sequences of events is the set of all possible event sequences, i.e. $S_E = \{E_i\}$. Event sequences in an event sequence space are considered to be mutually exclusive, even though they may partially overlap, since they are assumed to be related to dynamics, which originate from a single initiating state of the process variables and change at each branching point. Branching point represents a time moment when dynamics changes due to possible occurrence of event. At each branching time moment the dynamics may have one or more branches, which correspond to the following dynamics, depending on previous dynamics and possible occurrence of different events. Scenario s_i is a simplified representation of sequences with some common features. These features as example may concern the classification of events or/and separation according to timing of events. The sequences belonging to a same scenario are therefore considered to be similar, to the extent that they share the features implied by the scenario. Rare scenario from definition point of view is related to the rareness of sequences.

The practical simulation of sequences asks for the introduction of some kind of simulation approach. In the case of stimulus driven dynamics, the proposed approach considering discrete time moments restricts the amount of possible branching causes and branching points in time. After the occurrence of an initiating event, the related evolution laws of the process variables are considered. The corresponding deterministic process evolution defines a branch, from which all stimuli with related events causing the system to branch off are accounted for at those user-specified discrete time intervals. The same branching process is carried on until considered process variables reach final absorbing end states (consequence expressed as a damage state or as a steady safe situation). The frequency of all

sequences of dynamics can be calculated as they develop in the simulation.

3. Scenarios development

3.1. Issue of uncertainty treatment

The part of uncertainty related to PSA can be considered as a spread or distribution in the value of the risk estimate (frequency). Classically the spread in the risk estimate is related to the spread in the parameters of the probabilistic model used to estimate the risk R (Kaplan & Garrick 1981 [8]). As parameters usually are considered p_i - the probability of a risky i^{th} scenario, and c_i - the consequence or evaluation measure of that i^{th} scenario leading to damage.

Considering the frequency f_i with which the occurrence of scenario S_i is not exactly known, the state of knowledge about this frequency f_i can be expressed with a probability $p_i(f_i)$ using the probability density function for the frequency f_i, of the i^{th} scenario. Similarly, if there is uncertainty u_i in the i^{th} consequence estimation it can be expressed as $u_i(c_i)$, $i = 1, 2, \ldots, N$, where N is total amount of scenarios.

However, in addition to uncertain parameters of model, another cause of uncertainty may arise from the incompleteness, i.e. from the incomplete modeling or data used in the probabilistic model itself or in the analyses used to derive the model. The uncertainty in the model inputs may also affect the topology of the probabilistic model or the data and time dependence used to quantify it.

The completeness of the scenario list depends on the consideration of each scenario formation, i.e. on the way of grouping of sequences and assigning them correspondent frequencies and consequences. Thus the incompleteness in the amount z of scenarios may be related to the number N of the set of triplets for risk consideration:

$$R = \{<S_i, p_i(f_i), u_i(c_i)>\}, i = 1, 2, \ldots, z(N). \quad (6)$$

The level of risk conditioned by a scenario partially represents how rare and important this scenario is from the consequence point of view. However, in order to search for scenarios with unpredictable, but possibly severe, consequences, there is a need to generate and consider events and dynamics as well as related sequences, which can be very rare.

The technique to search for rare random events is evident and is related to traditional PSA; however there is also a concern for how to generate and to consider rare deterministic events, which are related to the dynamics itself, i.e. process variables and timing. Actually, this means that each scenario S_i is time-dependent and uncertainty related to this could be also considered during the scenario development.

Uncertainty related to PSA could be classified according to the uncertainty source: the frequency of events and the sequences of dynamics themselves (i.e. dynamics and timing in the process variables space) (Labeau & Izquierdo 2005). Taking into account that all sources of uncertainty are important, there is a need of such uncertainty analysis, which considers both sources and at the same time reflects the issue of model incompleteness.

On the basis of this classification, there is a need to note that in the case of the first source of uncertainty, changing the values related to the frequency of events occurrence (e.g. failure rate λ) will not create new branching situations, but it will affect the likelihood of already possible sequences and scenarios, without changes in the possible process variables evolutions and scenarios themselves. Conversely, a change in the value of an uncertain threshold related to a specific event creates a new dynamic trajectory in the process variables space.

Considering this classification, it is easy to conclude that the first source of uncertainty (i.e. fluctuations of failure or recovery rates, or of on-demand failure probabilities) can be propagated with no additional deterministic calculations, as all sequences in the process variables space keep valid. However, the second source of uncertainty, in principle, causes a continuum of additional scenarios with different timing, what requires to perform an approximation of the continuous event tree (Labeau et al. 2000 [10]). This uncertainty effect on the process variables evolutions and it should therefore be investigated separately in order to save computational resources and represent scenario-related uncertainty.

Taking into account the main features of scenarios, it can be seen that a simulation algorithm relevant for dynamic reliability problems and uncertainties should display the following characteristics:

- Search for rare scenarios under consideration;
- Represent uncertainty of considered scenarios.

3.2. Path of processes and sequence of dynamics

Once the general purpose of scenario development is established, the traditional event tree analysis can be derived as a particular case of this extended PSA approach with consideration of paths and sequences used to define these scenarios.

The consideration of a continuous event tree, in principle, containing an infinity of scenarios and sequences asks for the introduction of some kind of discrete approximations, in order to restrict the amount of possible branching points in time to discrete time moments.

After the occurrence of an initiating event, the corresponding evolution laws of the process variables are considered. The corresponding deterministic transient evolution defines a branch, from which all stimuli with related events causing the system to branch off are accounted for at user-specified discrete time intervals. The same branching process is carried on until the considered process variables reach one among possible final absorbing *end states* (consequence expressed as a damage state or as a steady safe situation). The frequency of all sequences of dynamics can be calculated as they develop, by aggregating the paths of processes that are defined below. For the *path of processes* the following is assumed:

- The process j_o starts at a steady point state $\overline{u}_o$ with given probability (usually very close to 1). Associated with the initial stimulus activation state J_o a random instantaneous initiating event with a known frequency triggers the process evolution in time.
- Process variables $\overline{x}$ can be reached from the initial point state $\overline{u}_o$ following the sequence of events through a compound path of processes.

Each of the path timing, i.e. time variables $\vec{\tau}_{j_n} = (\tau_{j_1}, \ldots, \tau_{j_n})$, coincides with some of the event time variables τ_i, ordered in increasing time and related to the evolution of the process variables $\overline{x}$ from the initial state $\overline{u}_o$, determining their sequence of occurrence up to the end state.

Under the previous assumptions, once the possible dynamic evolutions $\overline{g}_k(t, \overline{u}_k)$ and the initial state $\overline{u}_o$ are defined, the paths of processes possible may be determined including the timing. The set of paths with the same processes and consequences, but differing in the path timing will be called s *sequence of dynamics*.

The price to pay for the sequence of dynamics and paths of processes consideration is that there are many paths possible within a sequence of dynamics. Each path of processes represents a deterministic transient that may be simulated with available simulation technology. However, identifying the relevant stimulus that activates events and the paths within a sequence is a complex and key factor in the path of processes delineation and generation of sequences of dynamics. The *relevant paths* of processes, which reflect the sequence of dynamics, constitute a paths envelop that should also take into account the bounding of the safety objectives exceedance.

Considering the sequences of dynamics similarly the following is assumed:

- The system starts due to a random instantaneous initiating event in an initiating steady state of the process variables $\overline{u}_0$ with given frequency f_0. The initial state of the process variables $\overline{u}_0$ is associated to the initial status of activation of the stimuli denoted as $J_0 = J(j_0) = J(0)$ or (j_0, J_0), denoting j_0 the index of the first dynamics (i.e. which was entered in the initiating steady state of the process variables $\overline{u}_0$) and J_0 a vector of the activation status of stimuli J_k when entering the initiating dynamics ($k = j_0$).
- At any further time moment $t = \tau_n$ the process variables $\overline{x}(t)$ can be reached from the initiating steady state $\overline{u}_0$ starting from the initial time moment τ_0 going through n paths of processes (evolution of process variables) affected by the sequence of events (i.e. elapse of the delays from the stimuli activations).

In general, the paths of processes considering the initiating steady state of the process variables $\overline{u}_0$ and each initial state vector $\overline{u}_k$ in any dynamics k visited along the path can be described as follows:

$$\begin{aligned} &\overline{u}_0 = \overline{u}(J_0) = \overline{u}(J(0)) = \overline{x}_0 = \overline{x}_0(\tau_0) = \overline{x}_0(0), \\ &t^- \le \tau_0;\ \overline{x}_0(t) = \overline{g}_0\left(t - \tau_0, \overline{u}_0\right),\ \tau_0 < t \le \tau_1; \end{aligned} \tag{7}$$

$$\begin{aligned} &\overline{u}_k = \overline{u}(J_k) = \overline{x}_{k-1}(\tau_k) = \overline{g}_k\left(0, \overline{x}_{k-1}(\tau_k)\right); \\ &\overline{x}_k = \overline{x}_k(t) = \overline{g}_k\left(t - \tau_k, \overline{u}_k\right),\ \tau_k < t \le \tau_{k+1}, \\ &1 \le k < n; \end{aligned} \tag{8}$$

$$\begin{aligned} &\overline{u}_n = \overline{u}(J_n) = \overline{x}_{n-1}(\tau_n) = \overline{g}_n\left(0, \overline{x}_{n-1}(\tau_n)\right); \\ &\overline{x}(t) = \overline{x}_n = \overline{x}_n(t) = \overline{g}_n\left(0, \overline{u}_n\right),\ t = \tau_n. \end{aligned} \tag{9}$$

The occurrence times of the events in the vector $\vec{\tau}_n = (\tau_1, ..., \tau_n)$ are ordered in increasing time. They represent the starting moments of evolution of the physical variables $\overline{x}(t)$ from the initial state $\overline{u}_k$ in dynamics k, $1 \le k < n$.

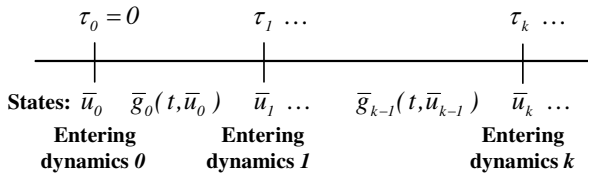


Figure 2. The sequence of dynamics vector.

Under the previous assumptions, once the initial states $\overline{u}_k^m$ from vector $\overline{u}_k$ with a specific activation status of stimuli $J_k^m \in J_k$ and possible dynamics $\overline{g}_k^r(t, u_k^r)$ from $\overline{g}_k(t, \overline{u}_k)$ are defined (with sojourn time t for each dynamics), the *sequence of dynamics* may be determined as the following set: $\{\overline{x}_k^r\} = \{\overline{g}_k^r(t, u_k^r)\}$, $1 \le r \le m$, $k \in \{0, 1, 2, ..., n\}$. The dynamics are represented in the paths of processes.

3.3. Scenarios and exceedance frequency

Each path of processes represents a deterministic transient that may be simulated with available simulation technology. The paths belonging to a sequence as well as the sequences belonging to a same scenario are considered to be similar, to the extent that they share the features (i.e. dynamics and timing) implied by the sequence and correspondingly by the scenario.

However, identifying the rare stimulus activations and the paths related to the different rare sequences is a complex and key factor in the process of uncertain scenario analysis. The scenario may constitute a sequences envelop (as for instance design basis envelops) that should also take into account the bounding of the consequence they may generate at the considered time moment τ_k.

When branching in an event tree is distributed in time (e.g. due to stochastic stimulus activation and delays), the problem of the combinatorial explosion of the number of sequences is well-known. In this case the considered branching time moments especially are frequent compared to only setpoint-based transitions (e.g. transitions between dynamics driven by stimuli without delays). In this situation, a large number of branching points could be defined on a branch of the event tree between two setpoints, with the possibility that scenarios due to these different branching points will lead to non-similar consequences from the safety and protection analysis standpoint.

In general, at each branching time moment τ_k the dynamics may have one or more branches, which correspond to the subsequent dynamics, depending on previous dynamics and the possible (non-) occurrence of different events. Consideration of only these time moments, which are related to any transitions, allows to be sure that there is at least one sequence when dynamics changes. However there are sequences of dynamics in which no changes occur at certain branching time moments and some subsequent dynamics in the sequence are similar.

For a simplified representation of such sequences, the concept of scenario can be used. S_j as *j scenario of dynamics* and the corresponding timing of events represent the set of sequences of different dynamics determined as the following set $S_j = \{\overline{s}_k^r\}$, $\{\overline{s}_k^r\} = \{\overline{g}_k^r(t, u_k^r)\}$ if $\forall j : \overline{s}_j^r \ne \overline{s}_{j+1}^r$, where $0 \le k \le n$, $1 \le r \le m$, $0 \le j < n$.

The initiating time moment τ_0 is related to dynamics $\overline{s}_0^1$. Other time moments τ_k important for scenario of dynamics are related to the initial states of dynamics $\overline{s}_k^r$ and can be determined from the vector $\vec{\tau}_n = (\tau_1, ..., \tau_n)$ according to the index k, where $0 \le k \le n$. The final dynamics in the scenario is related to the end state and the scenario consequence at this time moment τ_c, where $0 < c \le n$. In general, the final dynamics in the scenario can represent the steady state, which do not depend on time. However the final time moment considered in the scenario can be related to the occurrence of the scenario consequence.

It should be noted that taking into account that consequence is also time-dependent, and there is the possibility to consider the partial consequence (degradation or damage) after each dynamics. The time interval between changes of dynamics can also be related to the delays during which the deterministic process takes place and this case can be simulated using delays that do not need to relate to a stimulus.

In typical PSAs, operator delays are, to a large extent, ignored. However, the analysis of the event tree and sequences of events, which include operator action and are finally retained as failure sequences, can be revisited to obtain the so-called available time, or maximum time for operator response. It is possible that a shorter delay of action would result in the sequence be-

ing a success sequence, while a longer delay will be rare, but will produce different scenarios.

The paths of processes and sequences of dynamics that reflect the PSA approach allows us to use as the basic figure of merit the so-called *exceedance frequency*, which is the estimate of the failure probability related to the frequency of consequences (Alzbutas et al. 2007 [1]).

4. Conclusions

Considering the drawbacks in terms of representation of dependability and uncertainty and in terms of completeness of scenarios development process the concept of stimulated dynamics and delayed system was applied for PSA.

The delayed system consideration allows a systematic development and analysis of dependable and uncertain scenarios for PSA. It is applicable to search for relevant sequences and rare scenarios under consideration in PSA as well as represent and analyze uncertainty of certain scenarios.

Considering the expected outputs of a PSA the analysis based on stimulated dynamics and delayed system provide a new way of grouping sequences and scenarios according to their timing and consequences. The timing and dependability was shown as an important factor and the main source of scenarios uncertainty that can significantly influence PSA results.

5. References

[1] Alzbutas R. et al. “Application of stimulated dynamics to probabilistic safety assessment”, *Proceedings of ESREL: Safety and Reliability Conference 2*, 2007, pp. 1027-1034.

[2] Alzbutas R. and Janilionis V. “Aggregate simulation of stimulated dynamics for reliability analysis”, *Proceedings of ESREL: Safety and Reliability Conference 2*, 2007, pp. 1035-1041.

[3] Alzbutas, R. and Janilionis, V. **“**Determination and Simulation of Stimulated Dynamics”, *Science Works of Lithuanian Mathematicians Association 46*, 2006, pp. 321-327.

[4] Devooght J. “Dynamic Reliability”, *Advances in Nuclear Sciences and Technology 25*, 1997, pp. 215-278.

[5] Devooght, J. and Smidts, C. “Probabilistic Dynamics as a Tool for Dynamic PSA”, *Reliability Engineering and System Safety 52 (3)*, 1996, pp. 185-196.

[6] Izquierdo, J.M. and Labeau, P.E. “The stimulus-driven theory of probabilistic dynamics as framework for probabilistic safety assessment”, *Proc. of international conference PSAM 7 – ESREL’04 (2)*, 2004, pp. 687-693.

[7] Izquierdo, J.M. et al. “Relationship Between Probabilistic Dynamics and Event Trees”, *Reliability Engineering and System Safety 52*, 1996, pp. 197-209.

[8] Kaplan S. and Garrick J. “On the quantitative definition of risk”, *Risk Analysis 1*, 1981, pp. 11-27.

[9] Labeau, P.E. and Izquierdo, J.M. “Modeling PSA Problems-I: The Stimulus-Driven Theory of Probabilistic Dynamics”, *Nuclear Science and Engineering 150*, 2005, pp.1-25.

[10] Labeau, P.E. et al. “Dynamic Reliability: Towards an Integrated Platform for Probabilistic Risk Assessment”, *Reliability Engineering and System Safety 68*, 2000, pp. 219-254.

Dependability Modeling and Evaluation of an Automated Highway System

Ossama Hamouda, Mohamed Kaâniche, and Karama Kanoun
LAAS-CNRS; Université de Toulouse, 7, Avenue du Colonel Roche, F-31077 Toulouse, France
{firstname.lastname}@laas.fr

1. Introduction

Traffic congestion is increasingly growing, especially in urban areas. One of the solutions for this problem is automated traffic. Many research programs have been carried out or are currently underway to implement automated road or highway systems, based on automatically controlled platoons of vehicles. The aim is to improve the flow and the safety of traffic by reducing accidents, while reducing fuel consumption and pollution [1].

Automatic collaborative driving systems make use of inter-vehicles or vehicles-to-fixed infrastructure communications to autonomously guide cooperative vehicles on an Automated Highway System, e.g., for the aim of sharing the necessary information to ensure safe driving. The expected gains from automatic driving are the target of many research studies in this domain. However, to the best of our knowledge there was no previous work dedicated particularly to the dependability modeling and evaluation of such systems.

2. Context and Objectives

We address the evaluation of quantitative measures for characterizing the dependability in the context of the Automated Highway System, based on platoons in which vehicles are driven by more or less autonomous agents, interacting in a multi-agent environment [1]. Our work aims at developing evaluation methods and models that make it possible to analyze the safety of such platooning applications implemented in a mobile context with ad-hoc networks. Several phenomena such as accidental faults occurrence, vehicles mobility, frequent communication disconnection, are taken into consideration. This problem is challenging in the domain of automated highway applications and systems implemented in ad-hoc networks.

The developed models and measures are aimed at providing support to the designers for the selection and analysis of candidate architectures that are well suited to fulfill the dependability requirements of such a system. We consider as a case study the architectures developed in the framework of a project of intelligent transport systems (*Partners for Advanced Transit and Highways* "PATH" [2]), at Berkeley University. These architectures implement automatic maneuvers to ensure the platoons safety in the presence of different types of failure modes affecting the vehicles, their environment, and the inter-vehicle communication. The objective of our work is to propose a methodology based on analytical modeling techniques such as Markov chains and generalized stochastic Petri nets (GSPN) allowing us to model these maneuvers and evaluate their impact on the platoons safety.

3. PATH Architecture

The PATH research program proposed an hierarchical control architecture for controlling platoons of vehicles. This architecture improved the traffic throughput up to four times, while reinforcing safety. In this program, the vehicles platoons use lateral and longitudinal controllers (of positioning) in order to follow each others on roads equipped with magnetic marks. The automated vehicles are coordinated by both vehicle-to-vehicle and vehicle-to-fixed infrastructure communications, using a road-side infrastructure for traffic management purpose.

A *platoon* (*p*) is a series of vehicles that are moving in the same direction on a specified highway [2]. Each platoon contains a number of vehicles directed with one *leader* that is the first car of the platoon. The vehicles that follow the leader are called *followers*. A platoon that contains one vehicle is called free agent. Figure 1 shows three platoons: *p1* that contains three vehicles, a leader and two followers, *p2* an adjacent platoon, and *p3* an example of free agent.

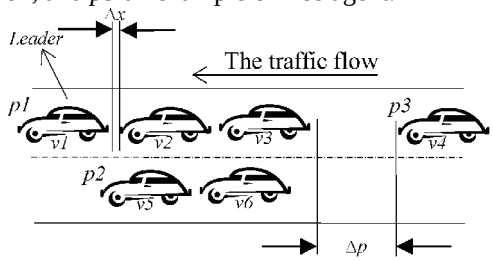


Figure 1: Context of a platooning application

The main maneuvers are: splitting of a platoon, merging of platoons, or making a vehicle exit or enter the platoon. In particular, in case of a failure affecting

a vehicle in the platoon, the maneuvers allow the vehicle to leave its platoon without any hazard, for the purpose of continuously running the platoon without any problem [3].

For starting a maneuver, the faulty vehicle communicates with its platoon's leader that determines the exit strategy and ensures the coordination for the maneuvers associated to this strategy. If the faulty vehicle is the leader, other maneuvers must be applied to allow the platoon vehicles to select the new leader. According to the failure mode, certain maneuvers may require a communication between the leaders of several platoons in the environment [3].

4. Methodology

Our objective is to evaluate dependability measures characterizing the safety of the automated highway system taking into account: i) the occurrence of different types of faults and failure modes affecting the vehicles or their communications, and ii) the strategies and maneuvers that have been designed to allow a faulty vehicle to leave a platoon without affecting the safety of the whole system including the adjacent platoons. The evaluation of these measures should take into consideration the different ways of vehicles coordination in the automated highway system.

Two main steps can be distinguished. The first step consists in identifying representative platooning scenarios, the failure modes to be taken into account, and their severities and the maneuvers needed to ensure safe platooning. The second step is dedicated to the elaboration of the GSPN analytical models incorporating the information identified in the first step to quantify safety and dependability measures.

As illustrated in Table 1 different failure modes with different levels of severities can be distinguished. The corresponding maneuvers to be applied to ensure the safety of the platoon are also identified. When multiple failure modes occur or when a maneuver fails, the maneuver with the highest priority is applied (see Figure 2). The success of a given maneuver depends on several factors, e.g., the states of the vehicles involved in the platoon and the density of the traffic.

The dependability modeling consists in the elaboration of GSPNs describing the behaviors of each vehicle involved in the considered platooning scenarios as well as the dependencies resulting from interactions between vehicles inside a platoon and between platoons during the execution of the safety maneuvers. One of the critical aspects to be addressed during modeling is to master the complexity of the models at the model construction and processing levels and to take into account the dynamicity of the vehicles. Hierarchical compositional approaches are currently investigated to address this problem.

Table 1: Examples of failure modes and maneuvers

Failure mode	Severity	Maneuver
FM1 (No Breaks)	A3	Aided Stop (AS)
FM2 (Inability to sense vehicles in adjacent lanes)	A2	Crash Stop (CS)
FM3 (Inter-vehicle communication failure)	A1	Gentle Stop (GS)
FM4 (Transmission failure)	B/B2	Take Immediate Exit-Escorted (TIE-E)
FM5 (acceleration controller down)	B2	Take Immediate Exit (TIE)
FM6 (communication between link and network layers down)	C	Take Immediate Exit-Normal (TIE-N)

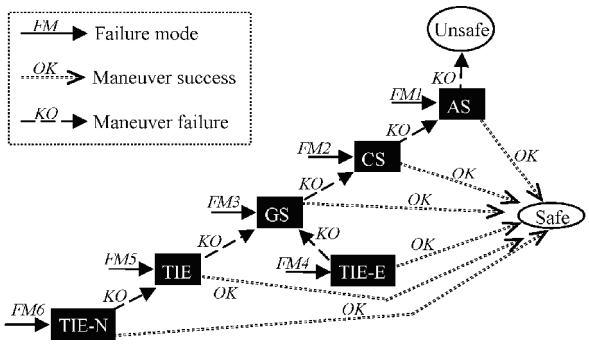


Figure 2: Failures modes, maneuvers, safety impact

5. Ongoing Work

Ongoing work is focused on the elaboration of the GSPN modeling framework considering platooning scenarios involving a limited number of vehicles and platoons. Sensitivity analyses will be performed to compare different communication and coordination schemes between the vehicles: e.g., centralized vs. decentralized inter-platoon or intra-platoon communication. Particular attention is also devoted to the investigation of efficient means to take into account vehicles mobility in the models.

Acknowledgment: This work was partially supported by the HIDENETs project *(HIghly DEpendable ip-based NETworks and Services)* http://www.hidenets.aau.dk/ (EU-IST- 26979).

6. References

[1] S. Hallé, "Automated Highway Systems: Platoons of Vehicles Viewed as a Multiagent System, M.Sc., "Faculté des études supérieures de l'Université Laval. Québec, 2005.

[2] P. Varaiya, "Smart Cars on Smart Roads: Problems of Control," IEEE Transaction on Automatic Control, vol. 38 No. 2, pp. 195-207, Feb. 1993.

[3] D. N. Godbole et al. A. Deshpande, and A. E. Lindsey, "Towards a Fault Tolerant AHS Design Part II: Design and verification of communication protocols," Institute of Transportation Studies, University of California, Berkeley, *paper* UCB-ITS-PRR-96-15 1996.

Session 3B: Student Forum

EDCC-7

Operating System Services for Recovering Errant Applications

Raul Barbosa
Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Göteborg, Sweden
rbarbosa@ce.chalmers.se

Abstract

Operating systems often manage critical infrastructures where failures can have serious consequences. This raises great concerns about their robustness. From the user perspective, it is the service delivered by host applications that needs to be dependable. Operating systems should therefore provide comprehensive error detection and recovery services to those applications, so that the system as a whole can be dependable and secure. This paper addresses the recovery flow that takes place after an application error is detected. The goal is to combine existing techniques into a set of operating system services that support application recovery both from software and hardware errors. We describe a prototype system where these services are currently being implemented and outline how we intend to experimentally evaluate them.

1 Introduction

Designing robust operating systems has been the goal of numerous research endeavours since the late 1950s, when the first mainframe computers were born. Since then, operating systems have managed critical infrastructures ranging from server rooms to embedded devices, as well as crucial user information on desktop computers. Given that a failure of such computers can have serious consequences, the operating systems must be reliable in the presence of faults. Moreover, they should also provide comprehensive error detection and recovery services to hosted applications, so that the system as a whole can be dependable and secure.

Computer systems are affected by faults of diverse origins. Software defects – introduced during development – can cause errors in the applications and in the operating system itself. For this reason, a large body of research has been devoted to preventing and removing software defects; this is, however, a complex problem lacking perfect solutions. Thus, operating systems have to cope with the existence and occasional activation of software faults, *e.g.*, when a buggy line of device driver code is executed. Another important source of errors are hardware faults – mostly of physical nature – due to environmental factors, aging, etc. These are now more concerning than ever, largely due to modern manufacturing processes and the associated variability and degradation of microprocessors [3, 13, 17].

This paper is chiefly concerned with detecting and recovering errant applications. Ultimately, from the user perspective, it is the service delivered by the applications that needs to be dependable. Recovering applications (in addition to isolating them) is therefore key to ensuring sustainable delivery of service. Accordingly, we should judge the dependability of an operating system not only for its resilience to failures but also for the services it provides to hosted applications with regards to error detection and recovery.

To this end, operating systems can make use of a vast multitude of techniques available in the literature. In this paper we address only those targeting hardware and software faults of non-malicious nature. Our goal is to combine several such techniques into a set of operating system services that detect application errors and support their recovery.

The first obstacle is that the recovery flow usually depends on whether the error was caused by hardware or software. This is generally hard to diagnose. Consider an example where the memory management hardware raises an exception indicating an erroneous memory access; it may have been caused by a missing pointer initialization in the software but also by corruption of a pointer due to a hardware fault. In this case it is difficult to choose, for instance, whether to rollback and retry the operation or to transfer control to a user-mode exception handler.

Let us assume that application errors can be detected by the operating system even though their cause is unknown. Our proposal is to consider at first that an error was caused by a transient hardware fault. To cope with these, we checkpoint applications frequently and rollback to a previous state upon error detection. If the cause was indeed a hard-

ware fault, there's a high chance that it will vanish. Otherwise, if an error is detected again before another checkpoint is made, we must assume that it was caused by a software fault. At this point we transfer control to an application-specific exception handler where the designer can decide what should be done. If no such handler is defined, the operating system may simply restart the application – this is often sufficient to deal with rarely activated bugs.

We are developing a prototype system which we intend to use for experimentally assessing various techniques aimed at building robust operating systems. The remainder of this paper details the design principles of the prototype and outlines the experimental evaluation. We begin by surveying related research in Section 2. Error detection is discussed in Section 3 and Section 4 elaborates on the strategy for recovering applications. The experimental evaluation through fault injection is discussed in Section 5. Lastly, we summarize the main conclusions of the paper in Section 6.

2 Related Research

A great deal of work has been dedicated to ensuring that operating systems are resilient to internal failures. In this context, kernel extensions such as device drivers are usually identified as the major source of problems. The *microkernel* approach attempts to solve this issue elegantly by isolating kernel extensions in user-mode, where fault containment can be more easily achieved. Herder *et al.* [6] use this design principle in the Minix operating system. There is a price to pay for the increased reliability: obtaining an operating system service often involves full context switching and additional data copying. Tanenbaum *et al.* [15] point out that this performance penalty is worth the trade-off in systems where reliability is the main concern.

Swift *et al.* [14] use a different strategy to tackle the same problem. Their approach, implemented in Nooks, uses the more common *monolithic* kernel structure where extensions run in kernel-mode. It should be emphasized that kernel-mode instructions access main memory through the memory management hardware – just like user-mode instructions. The difference is that user-mode execution has restricted access to privileged registers and instructions. If we abstract from malicious faults, device drivers can be isolated by marking unnecessary pages as read-only during their execution. The authors make use of this feature to implement *lightweight protection domains*. Additionally, they propose the usage of wrappers to monitor control-flow between the drivers and the kernel.

A different approach is taken by Hunt *et al.* in designing the Singularity operating system [7, 8]. The core of Singularity is a microkernel written almost entirely in Sing# – a type-safe programming language with low-level constructs. Instead of relying on isolation through hardware protection, device drivers and applications are executed in the context of *software-isolated processes*. Here, the fundamental principle is using static software verification to assure that untrusted components are unable to access memory outside their context. Moreover, interaction among programs takes place via contract-based channels that can be statically checked; and the programs themselves come with a manifest allowing static and dynamic verification of key properties. The emphasis is thus on fault prevention rather than fault tolerance.

When it comes to detecting application errors, operating systems make use of techniques such as memory protection and watchdog timers to ensure spatial and temporal isolation. Under these circumstances, faults may still propagate via the system call mechanism – the door used by applications to request operating system services and to reach the device drivers. Peterson *et al.* [11], Provos [12] and many others propose the usage of *sandboxing* as the means to protect the system call mechanism. This technique consists of interposing the access to system calls with a filter that enforces a given policy. In practice, this serves as an accurate error detection mechanism.

After an error is detected, the recovery process is often left to the application designer or the operating system simply terminates the application. As we described before, we adopt the well-known checkpoint and rollback method to recover applications from transient hardware failures. This method can also be applied directly by the hardware. Wang and Patel [16] use microarchitecture-level checkpointing and define a set of error symptoms which trigger the processor to return to a previous state. Since we assume a generic hardware platform, we opt for implementing checkpointing as an operating system service.

The effectiveness of a rollback recovery depends substantially on the checkpointing strategy. If checkpoints are too frequent, their overhead is too high; if they are too sparse in time, recoveries may require too much recomputation. The work by Ling *et al.* [10], among others, addresses the checkpointing strategy problem. They offer a mathematical model for optimal checkpointing under time-varying failure rates. Results of this kind are an important input to our design, as they provide the means to choose the optimal checkpointing strategy.

3 SECERN: An Extension to MicroC/OS-II

A current trend in the design of embedded systems is to allow multiple applications to share a common hardware platform. The goal is to avoid using dedicated microcontrollers in order to achieve scalability as the number of functions grows. Initiatives such as the standard interface for avionics applications, defined by the ARINC 653 specification [2], aim at developing the critical infrastructures

and operating systems that support this level of integration. Since these initiatives target safety-critical systems, a fundamental concern is to ensure that resource sharing can be accomplished in a safe and reliable manner.

We are developing a prototype system intended for experimentally assessing various techniques for building robust operating systems. The prototype is based on the MicroC/OS-II real-time kernel, designed by Labrosse [9]. The kernel's source code is very well documented and freely available for academic purposes, making it amenable to extension.

MicroC/OS-II lacks support for isolating applications from one another and from the operating system. To solve this, we began by extending the kernel with memory protection [1] using a Memory Management Unit (MMU). This was accomplished by making the distinction between processes and threads, where each process owns a private address space that contains one or more threads of execution. Memory allocation is done statically by the linker – a common design decision in embedded systems.

A distinguishing feature of our implementation is that the Translation Lookaside Buffer (TLB) – a small cache that speeds up address translation – is updated during context switch. Our approach is to update the TLB with the pages that belong to an application before running that application. Thus, we avoid handling TLB-miss interrupts during the normal execution. This, in turn, makes it simpler to determine the worst-case response time of programs (a very important issue in real-time systems).

The extended version of the operating system runs on a computer board featuring a Freescale MPC5554 microprocessor based on the PowerPC architecture. On this processor, the time needed for a full context switch without updating any TLB entries is 10 μs. Considering a typical embedded application with 4 pages of memory, it takes 31 μs for context switching; with 8 pages it takes 53 μs. However, this increased time is not a penalty, as handling TLB misses would be more expensive in the worst case and less deterministic.

3.1 Error Detection in SECERN

In addition to memory protection, our goal is to further develop the operating system extension – denominated SECERN – by combining several error detection and recovery techniques. Some of those techniques are based on the literature surveyed in Section 2, adapted to suit the performance needs of embedded applications. Below, we describe the error detection mechanisms that are implemented in the current version of SECERN.

Memory protection. Each process has a private memory space statically allocated by the linker. Accessing memory outside this address space raises a CPU exception that is handled by SECERN.

Processor exceptions. All other CPU exceptions are handled by SECERN as well. Many are classified as *recoverable*, meaning that they were caused by an application error and the remaining parts of the system are intact. In such cases, application recovery is triggered.

Unauthorized system calls. Our extension supplies the ID of the calling process through a kernel structure that can be used to filter erroneous accesses to the system call. The ID can be checked by the system call's exception handler in order to enforce a given access policy.

Invalid system call parameters. This mechanism uses the caller ID principle targeting the device drivers. Any kernel extension can make sanity checks on the parameters passed by the caller. An example may be a communication driver that checks the size of a buffer and reports an error of the caller application.

Task deadline missed. In SECERN, a thread is expected to release the processor before its deadline. A scenario where this does not happen is considered an error. We note that not all tasks are required to have a deadline (*e.g.*, background tasks). These should execute with low priority to ensure that they cannot interfere with other programs.

Application-specific checks. The application designer is able to report errors to the operating system. In this case, the operating system will attempt rollback recovery as if the error had been detected by any of the previous mechanisms.

4 Recovering Errant Applications

The main contribution of this paper is the recovery flow that takes place after an application error is detected by one of the above-described mechanisms. The goal is to provide a comprehensive method that allows applications to be recovered both from hardware and software errors. To make this possible, one must diagnose the cause of each error, so that the appropriate recovery technique can be invoked.

Our proposal is to checkpoint the applications frequently and attempt rollback recovery upon error detection. There is a high chance that the error will vanish if it was caused by a transient hardware fault. However, it may also be the case that an error is detected again (before the next checkpoint). In this case, we diagnose it as being caused by software. The operating system can be configured to choose, at that point, one of two options: (i) restart the task; (ii) transfer control to an application-specific exception handler. In the

second case, the designer may decide, for example, to implement a backup routine that produces a degraded result which is known to be safe.

One must also consider the case when an error is detected a third time, *i.e.*, when both recovery attempts fail. Such cases may be caused by permanent hardware faults which prevent an application from executing correctly. Under such circumstances, the application is terminated in order to ensure fail-silence and to constrain resource consumption.

4.1 Checkpointing Strategy

The checkpointing strategy was identified earlier in the paper as determinant for the performance of the recovery process. Essentially, we need to know *what* part of the state should be saved and *when* it should be saved. Regarding what to save, we intend to begin by implementing an agnostic strategy, where the entire address space and processor registers in the context of an application are saved. There are more efficient approaches but our first goal is to evaluate the effectiveness of our proposed method, rather than obtaining the best performance.

Regarding the moment when checkpoints should be made, we propose to have a user-mode thread, with permission to read everywhere, copying the state of applications when they are idle. This way we can make use of the available slack time to increase the dependability of the system. To achieve this, the operating system should provide a service which applications use to request checkpoints. Those requests are queued and serviced by the low-priority checkpointing thread.

To ensure a checkpoint's consistency, we must guarantee that it is completed within one idle period of the application being checkpointed. The concern here is that the checkpointing thread can be preempted by an application while saving its state. When this occurs, the checkpointing thread must restart copying the application's state, since it may have been modified during the execution of the application. For this reason, the context switching routine should notify the checkpointing thread whenever an application is executed. This can be done by updating a single value in an array containing one element for each application, thereby keeping the overhead minimal.

4.2 An Alternative to Checkpointing

There are other techniques for masking hardware errors that can be used instead of checkpoint and rollback. We consider using *time redundancy* by executing the same task a second time when an error is detected. This method is appropriate if we consider the classic task models from which the *rate monotonic* and *earliest deadline first* results were derived. Here, tasks are issued periodically with an input and must produce their output by a given deadline. As long as task deadlines are met, we can use time-redundant execution to recover from transient failures.

From a practical viewpoint, running the task a second time upon error detection is equivalent to restoring a pseudo-checkpoint at the beginning of the task's execution. The advantage is that it does not require saving the state of applications, which may be a time-consuming activity, depending on the implementation. This method is appropriate for *stateless* tasks where each iteration is only dependent on its input. Moreover, we can also use this technique for tasks that depend on an external state that they cannot update (*e.g.*, the current operation mode); in this case we can view the external state as part of the input to the task.

5 Evaluation Outline

In this section we outline an experimental evaluation of the techniques described throughout the paper. Our goal is to assess, using fault injection, the robustness of the SECERN extension with regards to error detection and recovery. First, we are interested in ensuring that faults in an application do not propagate to the operating system nor to other applications (*i.e.*, isolation). Second, the proposed recovery flow must be validated and evaluated to ensure its usefulness and practicality.

With this in mind, we have developed a fault injection tool, for our experimental setup, that can inject faults and monitor the outcomes. The evaluation should use two independent processes receiving input and producing output using simple communication primitives. The tool injects faults in one of the applications and intercepts the results (of both applications) in the form of value/time pairs. The results are then classified regarding their correctness/timeliness and the activation of error detection mechanisms is monitored.

We consider injecting both hardware and software faults in our experiments. Hardware faults are emulated through bit-flips in registers and memory. Regarding software faults, we intend to use the most representative operators found by Durães and Madeira in their field study of open-source programs [4]. It should be noted that we must inject software faults manually in the source code of applications, as we have no automatic tool (*e.g.*, [5]) for this purpose. Nevertheless, we still depend on the fault injection tool for monitoring and classifying the outcome of each fault injection experiment.

5.1 Research Questions

We have identified a number of research questions which we expect to address during the experimental evaluation. The most important ones are:

How well does SECERN handle faults? In general, we're interested in understanding how robust operating systems can be built (focusing on embedded applications). To this end, fault injection is a good approach to testing, for instance, the coverage of the error detection mechanisms. A possible outcome would be to identify vulnerabilities in our extension.

How effective is each error detection mechanism? Each technique has a certain probability of detecting hardware and software errors. When several techniques are combined, there is a chance that some of them overlap and that a given technique is not effective. Answering this question allows us to make an informed decision on which set of mechanisms is the best.

Does memory protection ensure isolation? One can assume that the hardware itself has no design faults. However, there may be hidden faults in the tightly coupled software that manages that hardware. The goal here is to evaluate the isolation of applications upon hardware and software failures.

What is the impact of hardware faults? It is known that a hardware fault can potentially disrupt an entire hardware unit. However, the operating system only executes 5% of the time for a typical workload [9]. During the remaining time the processor is either idle or running applications. It is therefore likely that many transient hardware faults will only affect the context of a single application, much like software faults. We intend to estimate how likely this is.

Is the recovery flow effective and efficient? In this paper we proposed a recovery flow that addresses both hardware and software faults. Using fault injection we can evaluate the benefits of using this approach, the overhead it imposes and the latency of the recovery process.

6 Conclusion

This paper addresses the problem of recovering errant applications. We argue that an operating system's dependability should be judged not only for its resilience to failures but also for the services it provides to hosted applications with regards to error detection and recovery. The reason for this is that the applications are responsible for providing service to the system users. Thus, isolating faulty applications from one another (and from the operating system) is fundamental, but recovering them is also essential to ensure sustainable delivery of service.

Effective error detection is the premise to a successful recovery. We describe the error detection mechanisms implemented in SECERN – our extension to MicroC/OS-II which aims at providing robust fault containment around software components. This extension provides, among other mechanisms, hardware-supported memory protection. This feature is intended to prevent faults in one component from propagating to other components through the memory spaces. Memory protection is uncommon in embedded systems due to, among other factors, its impact on the worst-case response time of applications. However, we have opted for inserting, in the TLB, the page table entries of a process while context switching to that process. In this way, we avoid handling TLB-miss interrupts during the execution of applications, thereby eliminating the unpredictability of memory accesses.

In this paper we propose a recovery flow that takes place after an application error is detected. The method consists in assuming, at first, that an error is caused by a hardware fault and attempting rollback recovery. If the error is detected again (*i.e.*, the rollback recovery is unsuccessful), its cause is diagnosed to be a software fault. In this case, the principle is to allow the application designer to choose from a set of options. One option is to transfer the control to an application-specific exception handler where a different version of the software is executed; another option is to simply restart the application. Indeed, one of the principles adopted in this paper is to provide the application designer with many different options for recovery, so that the best choice can be made depending on the needs of each system.

Lastly, we outline an experimental evaluation of the proposed operating system extension. The goal is to assess, using fault injection, various techniques aimed at building robust operating systems. To clarify the desired outcomes of this evaluation, we identify a number of important research questions. We intend to complete the implementation of the recovery flow and address those research questions in the near future.

References

[1] J. Alçada, R. Barbosa, and J. Karlsson. Memory protection in a real-time kernel (fast abstract). In *Proceedings of the 6th European Dependable Computing Conference (EDCC-6), Supplemental Volume, Coimbra, Portugal*, Oct. 2006.

[2] ARINC Incorporated. ARINC specification 653-1: Avionics application software standard interface, Oct. 2003.

[3] S. Borkar. Designing reliable systems from unreliable components: The challenges of transistor variability and degradation. *IEEE Micro*, 25(6):10–16, Nov.-Dec. 2005.

[4] J. Durães and H. Madeira. Definition of software fault emulation operators: A field data study. In *Proceedings of the 2003 International Conference on Dependable Systems and Networks (DSN 2003), San Francisco, CA, USA*, pages 105–114, June 2003.

[5] J. Durães and H. Madeira. Emulation of software faults: A field data study and a practical approach. *IEEE Transactions on Software Engineering*, 32(11):849–867, Nov. 2006.

[6] J. N. Herder, H. Bos, B. Gras, P. Homburg, and A. S. Tanenbaum. Failure resilience for device drivers. In *Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2007), Edinburgh, UK*, pages 41–50, June 2007.

[7] G. C. Hunt and J. R. Larus. Singularity: Rethinking the software stack. *ACM SIGOPS Operating Systems Review*, 41(2):37–49, Apr. 2007.

[8] G. C. Hunt, J. R. Larus, M. Abadi, M. Aiken, P. Barham, M. Fähndrich, C. Hawblitzel, O. Hodson, S. Levi, N. Murphy, B. Steensgaard, D. Tarditi, T. Wobber, and B. D. Zill. An overview of the Singularity project. Technical Report MSR-TR-2005-135, Microsoft Research, Oct. 2005.

[9] J. J. Labrosse. *MicroC/OS-II: The Real-Time Kernel*. CMP Books, second edition, 2002.

[10] Y. Ling, J. Mi, and X. Lin. A variational calculus approach to optimal checkpoint placement. *IEEE Transactions on Computers*, 50(7):699–708, July 2001.

[11] D. S. Peterson, M. Bishop, and R. Pandey. A flexible containment mechanism for executing untrusted code. In *Proceedings of the 11th USENIX Security Symposium, San Francisco, CA, USA*, pages 207–225, Aug. 2002.

[12] N. Provos. Improving host security with system call policies. In *Proceedings of the 12th USENIX Security Symposium, Washington, DC, USA*, pages 257–272, Aug. 2003.

[13] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers. The impact of technology scaling on lifetime reliability. In *Proceedings of the 2004 International Conference on Dependable Systems and Networks (DSN 2004), Florence, Italy*, pages 177–186, June-July 2004.

[14] M. M. Swift, B. N. Bershad, and H. M. Levy. Improving the reliability of commodity operating systems. *ACM Transactions on Computer Systems*, 23(1):77–110, Feb. 2005.

[15] A. S. Tanenbaum, J. N. Herder, and H. Bos. Can we make operating systems reliable and secure? *IEEE Computer*, 39(5):44–51, May 2006.

[16] N. J. Wang and S. J. Patel. ReStore: Symptom-based soft error detection in microprocessors. *IEEE Transactions on Dependable and Secure Computing*, 3(3):188–201, July-Sept. 2006.

[17] M. Zhang, T. M. Mak, J. Tschanz, K. S. Kim, N. Seifert, and D. Lu. Design for resilience to soft errors and variations. In *Proceedings of the 13th IEEE International On-Line Testing Symposium (IOLTS 2007), Crete, Greece*, pages 23–28, July 2007.

Fault-Tolerant Real-Time Scheduling using Chip Multiprocessor

Risat Mahmud Pathan
Department of Computer Science and Engineering
Chalmers University of Technology, SE-412 96, Göteborg, Sweden
risat@chalmers.se

Abstract

Failure to meet task deadline in safety critical real-time systems can be catastrophic. Moreover, fault tolerance is a crucial aspect of such systems if faults are likely. In this work, fault tolerance in real-time systems is proposed using time redundancy to mask at most F transient faults. Schedulability of a set of n preemptive real-time periodic tasks using Rate-Monotonic(RM) schedule in chip multiprocessor (CMP) is considered. Chip multiprocessor rather than uniprocessor is proposed to make more CPU time available before deadline of tasks. This paper addresses the issue of finding maximum number of tasks that can run in parallel at a particular time and also finds minimum number of processing cores required in a CMP, denoted by MinC, to make the real-time system task set schedulable. A real-time fault-tolerant algorithm FT-RT-CMP for scheduling tasks using MinC cores is also developed considering the worst case distribution of F transient faults.

1. Introduction

System with strict timing requirements are used in several applications like fly-by-wire, brake-by-wire, autopilot system and space shuttles, industrial process control and robots [1-3]. Some of these systems are considered as *hard real-time systems* where missing the deadline of a task can pose threat to human lives or environment. Moreover, when faults occur, recovery from the fault must be considered in such system. *Time redundancy* rather than *space redundancy* can be used due to cost, volume and space considerations. Time redundancy technique to mask faults at node level requires re-execution of tasks, known as Temporal Error Masking (TEM) [3-5]. Enough CPU time may not be available in the real-time schedule for such re-execution if the number of faults and task execution time is large. To make more CPU time available, an approach is made in this paper to achieve better schedulability using Rate-Monotonic(RM) scheduling for a set of n preemptive periodic tasks to tolerate a maximum of F transient faults using chip multi processor(CMP). In processor industry, the trend is now to build single chip multiprocessor(CMP) seeing the diminishing return from uniprocessor with higher transistor count [6-8]. In this paper, using such processor chip, with multiple cores having more computational power, scheduling of safety critical real-time application tasks to mask the effect of faults using time redundancy is proposed.

This paper in organized as follows: Section 2 provides the related work, section 3 presents the task model, and the inherent parallelism with such task model is discussed in Section 4. Task instances that can run in parallel at different cores of a CMP are defined formally in section 5. Section 6 presents algorithm *FT-RT-CMP* for scheduling tasks using CMP. Section 7 concludes this paper with a pointer to future work.

2. Related Work

In this work, I address the issue of tolerating transient faults that are temporary malfunctioning of the computing units. Such temporary malfunction can lead to an error in the system. The main source of transient faults is environmental disturbances like power fluctuations, electromagnetic interference and ionizing radiation by alpha particles. Several studies have shown that transient faults occur at much a higher rate than permanent faults [9-11]. In [9], measurements showed that transient faults are 30 times more frequent than permanent faults, while the work in [10] revealed that 83% of all faults were determined to be transient or intermittent. In some real-time systems such as satellites and space shuttles, transient faults occur at a much higher rate than in general purpose systems [11]. Such high occurrences of transient faults motivated me to the development of an approach to tolerate transient faults using time redundancy. Many approaches have

been taken to address the use of time redundancy to tolerate transient faults [3-5, 12-14]. The work in [3] evaluates a real-time kernel that employs TEM for brake-by-wire applications where correct results increased from 81% to 89%. In [5], Ramos-Thuel presented an algorithm for fault-recovery based on concept of slack stealing. The work in [12] presented a recovery scheme using re-execution in the event of single or multiple errors. In [13], a temporal-redundancy-based recovery technique that tolerates transient task failures where tasks have different constraints is presented. An appropriate schedulability analysis for fault tolerant systems is made where recoveries of tasks may be executed at higher priority levels [14]. In my work here the execution of recovery tasks due to errors run as the same priority of the task in which the error is detected.

Since time redundant execution requires much CPU time, the use of multiple cores single chip multi processor(CMP) is proposed in this work. Building more powerful uniprocessors with increasing transistor counts has ceased due to limited instruction level parallelism, increased wire delay and latency to main memory access. Now trend is to accommodate many cores in the same die area, called Chip multi processor [6-8, 15]. On applications with large grained thread-level parallelism the multiprocessor microarchitecture performs 50–100% better than the wide superscalar microarchitecture [6]. Niagara chip multiprocessor increases application performance by improving throughput [7]. To application software, a Niagara processor will appear as 32 discrete processors with the OS layer abstracting away the hardware sharing [7]. If such processor is used for real-time scheduling, a total of 32 tasks can be scheduled in parallel using the support from operating system.

The low use of CMP today is because converting today's uniprocessor programs into multiprocessor ones is difficult. But in Section 4 of this paper we will see inherent parallelism in the real-time periodic task is the best target for CMP. For such inherent parallelism, the CMP is much more promising because it is already partitioned into individual processing cores [8]. To harness the benefit of CMP, applications must expose their thread-level parallelism to the hardware. This can be done by decomposing a program into parallel "tasks" and allow an underlying software layer to schedule these tasks on different threads [15]. Inspired by this approach, in this work task scheduling in CMP is addressed considering the potential parallelism within real-time periodic task set. In Section 6, the number of minimum cores in a CMP, denoted by *MinC,* required to make a task set schedulable is determined. In the next section, the task model used in this wok is presented.

3. Task model

The task set consists of n tasks, $\Gamma = \{\tau_1, \tau_{2,\ldots} \tau_n\}$. Each task τ_i has a period T_i, and a relative deadline D_i is equal to task period T_i, worst case execution time C_i and priority P_i. The highest priority task has the lowest period. The length of the Planning Cycle (PC) in which the tasks repeat themselves iteratively is the least common multiple of all task periods.

$$PC = \text{L.C.M.}\{T_1, T_{2,\ldots,} T_n\}.$$

Within one *PC*, one or more instances of task τ_i will execute. Each *task instance* is denoted by τ_{ij} where j is the j^{th} instance of task τ_i. The set of tasks that get ready at time t is denoted by $RD(\Gamma,t)$ defined as:
$RD(\Gamma,t)=\{\tau_{ij} | \tau_i \in \Gamma$ and $t=(j-1)T_i$ for $j=1,2\ldots (PC/T_i)\}$.
Γ_{all} is defined as the set of all task instances within PC. That is, $\Gamma_{all}=\{\tau_{ij} | i=1,2,\ldots n$ and $j=1,2,\ldots.(PC/T_i)\}$. All time units used in this work is integer values.

In this work, temporal error masking (TEM) for fault tolerance is used as follows: when a task is released, two *primary copies* of the task instance are run first. If an error is detected by comparison, or by error detection mechanism, F more *extra/recovery copies* of the same task instance are run and a majority voting is made to mask at most F errors. Figure 1 demonstrate this for F=2 and for a single task τ_1 with period T_1=10 and C_1=2.

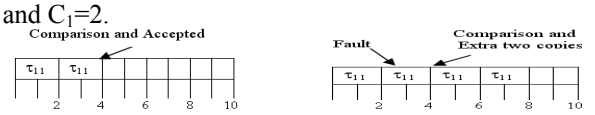


Figure 1. Fault free (left) , Fault Masking (right)

This paper is based on the assumption that, transient faults in different copies of the same task produce different outputs. As a result, the probability of having the same error in two primary copies is very small and error detection by comparison is possible. In next section, the inherent parallelism in tasks and how CMP could exploit such parallelism is discussed.

4. Task Level Parallelism

In Rate-Monotonic scheduling [16], the critical instance of the task sets is when all n tasks are released at time t=0. At time t task τ_i is released if $t=mT_i$ for some nonnegative integer m. When TEM is used, each of such primary copies of a task instance run twice in uniprocessor and can run in parallel in two cores of a CMP. Moreover, F more extra/recovery copies of a task instance are ready to run when error is detected. Each of the F extra/recovery copies can run in parallel in CMP if enough cores are available. These two scenarios show the inherent parallelism in application tasks of real-time system. In next subsection 4.1, how CMP can exploit such parallelism is demonstrated using an example.

4.1. Parallelism Exploitation by CMP

The inherent parallelism that can be exploited by CMP is demonstrated here using an example. Consider a real-time system with F=1 and two tasks τ_1 and τ_2 as in Figure 2 (left one). Also consider that, one of the two primary copies of τ_1 is in error. The Rate-Monotonic schedule is in Figure 2 (right one) with recovery copy running at t=2 to t=3.

τ_i	T_i	C_i
τ_1	3	1
τ_2	6	1

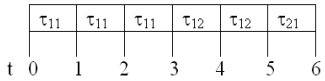


Figure 2. Task set (left), RM schedule (right)

The second instance of the first task τ_{12} (error free instance) finishes at t=5. The first instance of task τ_{21} does not have two time units with PC for execution of its two primary copies. So, the task set is not schedulable.

Considering the task level parallelism, the task model used in this work is an excellent target for CMP. Figure 3 shows two RM schedules for task set in Figure 2 for F=1 with two cores and three cores CMPs.

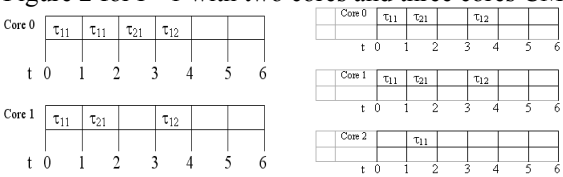


Figure 3: 2-Core (left) and 3-Core (right) rate monotonic schedule

In these CMPs schedules, the task τ_{21} is schedulable and other tasks have low response time and high slack is available that can be used to tolerate more faults or can be used to run other hard or soft aperiodic tasks.

Next question arises: how to generate such a schedule using CMP? How many tasks can run in parallel at any time instance t? In the next section, task instances that can run in parallel at time t are determined formally.

5. Parallel Task Instances

In the section 5.1 the set of primary copies of task instances, denoted by *PEX(t)*, that are ready to execute at time t in the fault free execution scenario is defined. In section 5.2 the set of extra/recovery copies of task instances, denoted by $FEX^F(t)$, that are ready to run at time t due to F faults is determined. In section 5.3 the set *EX* is defined combining the sets *PEX(t)* and $FEX^F(t)$ to find the set of primary and extra task instances that are ready at to run within PC due to F faults. The set *EX* is used to find the minimum number of cores, denoted by *MinC,* to schedule all tasks using CMP with the scheduling algorithm *FT-RT-CMP* is defined in section 6.

5.1 Primary Copies of Tasks at time t: PEX(t)

The set *PEX(t)* contains triplets (a, τ_{ij}, b) such that the at time a task instance τ_{ij} need b unit of execution time in the uniprocessor RM schedule in the fault free case (where only two primary copies of τ_{ij} run). That is, $PEX(t) \subseteq \{t\} \times \Gamma_{all} \times N$. For example, $PEX(1) = \{(0,\tau_{11},1), (0,\tau_{21},2)\}$ for task set in Figure 2.

Let the function $HP(PEX(t)) = \tau_{lk}$ such that τ_{lk} is the highest priority task in set *PEX(t)*. For example, $HP(PEX(1)) = \tau_{11}$ for the example task set in Figure 2. Lets formally define the function *PEX(t)* as follows:

$$PEX(t)=\{(t,\tau_{ij},2C_i) \mid \tau_{ij} \in RD(\Gamma,t)\} \quad \text{if } t=0 \text{ or}$$

$$\begin{cases} \text{if } t>0 \text{ and } t \neq PC \text{ and } PEX(t-1)=\varnothing \\ (PEX(t-1) - \{(a,\tau_{lk}, x)\}) \cup \{(a,\tau_{lk}, x-1)\} \\ \text{if } t=PC \text{ and } \tau_{lk}=HP(PEX(t-1)) \text{ and } x>1 \\ (PEX(t-1) - \{(a,\tau_{lk}, x)\}) \\ \text{if } t=PC \text{ and } \tau_{lk}=HP(PEX(t-1)) \text{ and } x=1 \\ (PEX(t-1) - \{(a,\tau_{lk}, x)\}) \cup \{(a,\tau_{lk}, x-1)\} \cup \\ \qquad \{(t,\tau_{ij},2C_i) \mid \tau_{ij} \in RD(\Gamma,t)\} \\ \text{if } t<PC \text{ and } \tau_{lk}=HP(PEX(t-1)) \text{ and } x>1 \\ (PEX(t-1) - \{(a,\tau_{lk}, x)\}) \cup \\ \qquad \{(t,\tau_{ij},2C_i) \mid \tau_{ij} \in RD(\Gamma,t)\} \\ \text{if } t<PC \text{ and } \tau_{lk}=HP(PEX(t-1)) \text{ and } x=1 \end{cases}$$

Let consider the following task set in Table 1 where the PC=14. The PEX(t) is shown in Table 2.

Table 1: Task Set

τ_i	T_i	C_i
τ_1	7	2
τ_2	14	1

Table 2: PEX(t) for task in Table 1

t	PEX(t)
0	{(0,τ_{11},4)(0,τ_{21},2)}
1	{(0,τ_{11},3)(0,τ_{21},2)}
2	{(0,τ_{11},2)(0,τ_{21},2)}
3	{(0,τ_{11},1)(0,τ_{21},2)}
4	{(0,τ_{21},2)}
5	{(0,τ_{21},1)}
7	{(7,τ_{12},4)}
8	{(7,τ_{12},3)}
9	{(7,τ_{12},2)}
10	{(7,τ_{12},1)}
6,11,12,13,14	{}

From Table 2, it is clear that at time t=0, two primary copies of each task can run in parallel if four cores are available. Using set *PEX*, the execution finishing time of two primary copies of a task can be determined. For example, the finishing time of τ_{11} is 4 in Table 2 since this is the earliest t at which τ_{11} disappears from PEX(t). At t=14, new tasks are released but PEX(14)={} since only tasks released within PC are considered. The following functions are defined to deal with the faulty case in next section:

fin(τ_{ij}) = The execution finishing time of the primary two copies of task τ_{ij}. So, ***fin***(τ_{11})=4

pre((τ_{ij}))=τ_{lk} such that τ_{lk} finished immediately before τ_{ij} in fault free execution. So, ***pre***(τ_{12})= τ_{21}

slack(t_1,t_2)= k where k is the number of free slots between time t_1 and t_2 in a schedule . So, ***slack***(6,7)=1.

5.2 Extra/Recovery Tasks at time t: $FEX^F(t)$

In this section, the worst case fault distribution for a maximum of *F* faults as in [4] is considered. Similar to PEX(t), the set $FEX^F(t)$ contains triplets (p, q, r) where at time *p*, the extra/recovery copy of task instance *q* still needs a of total *r* units of execution time due to faults. In the following equations, the constant F is used for the number of maximum faults and the variable *f* is used to signify *f* number of faults, where $f \leq F$, to deal with scenario where less than F faults occurs. These extra copies, in $FEX^F(t)$ at time *t*, can be derived from the extra copies that are ready to run at the finishing time of the primary task copies. When a task τ_{ij} finishes execution of both primary copies at $t=fin(\tau_{ij})$, there are two cases to consider for worst case *f* faults distribution. *Case1* (defined by $Q^1_{ij}(f)$): all *f* faults caused *f* errors have already occurred before $fin(\tau_{ij})$. *Case2* (defined by $Q^2_{ij}(f)$: all (*f*-1) faults and consequent (f-1) errors have occurred before $fin(\tau_{ij})$ and a new fault is detected at $t=fin(\tau_{ij})$. The proof for the worst case scenarios based on case 1 and case 2 can be proved using induction on the number of faults. The total processing time required for all tasks in set $Q^1_{ij}(f)$ and $Q^2_{ij}(f)$ are defined using functions $W^1_{ij}(f)$ and $W^2_{ij}(f)$ respectively later. Before that, the set of triplets of extra tasks that are ready to run at $t=fin(\tau_{ij})$ due to f faults is defined by FEX^f_{ij} as follows:

$$FEX^f_{ij} = \begin{cases} \varnothing & \text{if f=0} \\ \{(fin(\tau_{11}),\tau_{11}, F\times C_1)\} & \text{if ij=11} \\ Q^1_{ij}(f) & \text{if } W^1_{ij}(f) > W^2_{ij}(f) \\ Q^2_{ij}(f) & \text{if } W^2_{ij}(f) > W^1_{ij}(f) \\ Q^K_{ij}(f) & \text{if } W^1_{ij}(f)=W^2_{ij}(f) \text{ and } Q^K \text{ has higher priority task than in } Q^{K(\text{mod } 2)+1} \end{cases} \quad \ldots \quad \ldots \quad \ldots(I)$$

The sets $Q^1_{ij}(f)$ and $Q^2_{ij}(f)$, and the functions $W^1_{ij}(f)$ and $W^2_{ij}(f)$ for *case1* and *case 2* are defined as follows:

Case 1: This case deals with scenario where all *f* errors have already occurred in tasks that has finished execution before $t=fin(\tau_{ij})$. The set $Q^1_{ij}(f)$ contains triplets (t, b ,c) such that at time $t=fin(\tau_{ij})$, the extra copy of task *b* with execution time *c* are ready to execute. $Q^1_{ij}(f)$ is defined as follows:

$Q^1_{ij}(f) = SQ^s_{ij}(f)$ where $s=slack(fin(pre(\tau_{ij})), fin(\tau_{ij}))$ $\quad \ldots \quad \ldots \quad \ldots(II)$

$SQ^s_{ij}(f)$ is defined as follows:

$$SQ^s_{ij}(f)= \begin{cases} FEX^f_{pre(\tau ij)} & \text{if s=0} \\ SQ^{(s-1)}_{ij}(f) & \text{if } SQ^{(s-1)}_{ij}(f) =\varnothing \\ (SQ^{(s-1)}_{ij}(f) - \{(a,\tau_{lk}, x)\}) \cup \{(a,\tau_{lk}, x-1)\} & \text{if } \tau_{lk}=HP(SQ^{(s-1)}_{ij}(f)) \text{ and } x>1 \\ SQ^{(s-1)}_{ij}(f) - \{(a,\tau_{lk}, x)\} & \text{if } \tau_{lk}=HP(SQ^{(s-1)}_{ij}(f)) \text{ and } x=1 \end{cases} \quad \ldots \quad \ldots \quad \ldots(III)$$

The function $SQ^s_{ij}(f)$ selects the highest priority extra copy of tasks in $FEX^f_{pre(\tau ij)}$ and reduces the execution time based on the slack available between the finishing time of the current and the previously completed task.

Case 2: All the (f-1) faults have occurred before $t=fin(\tau_{ij})$ and a new fault has occurred in τ_{ij} at $t=fin(\tau_{ij})$and the triplets in $Q^2_{ij}(f)$ is defined as follows:

$Q^2_{ij}(f)= Q^1_{ij}(f\text{-}1) \cup \{(fin(\tau_{ij}),\tau_{ij}, F \times C_i)\}$ … …(IV)

The amount of extra workload at time t is the sum of all execution times of the triplets in $Q^1_{ij}(f)$ and $Q^2_{ij}(f)$ and is defined by $W^1_{ij}(f)$ and $W^2_{ij}(f)$:

$W^1_{ij}(f)= \Sigma X$ such that $(a,\tau_{lk}, X) \in Q^1_{ij}(f)$ … …(V)

$W^2_{ij}(f)= \Sigma X$ such that $(a,\tau_{lk}, X) \in Q^2_{ij}(f)$ … …(VI)

Now, the set of triplets in $FEX^F(t)$, as explained at the beginning of this section, is defined as follows:

$$FEX^F(t)= \begin{cases} \varnothing & \text{if t=0} \\ FEX^F_{ij} & \text{if t=fin } (\tau_{ij}) \\ FEX^S_{ij}(t) & \text{if } fin(\tau_{ij}) < t < fin(\tau_{lk}) \text{ and } pre(\tau_{lk})=\tau_{ij} \text{ and } s=slack(fin(\tau_{ij}),t) \\ (FEX^F(t\text{-}1) - \{(a,\tau_{lk}, x)\}) \cup \{(a,\tau_{lk}, x\text{-}1)\} & \text{if } \tau_{lk} =HP(FEX^F(t\text{-}1)) \text{ and } x>1 \text{ and } FEX^f(t\text{-}1)\neq\varnothing \text{ and } [t\text{-}1,t] \text{ is a free slot} \\ FEX^F(t\text{-}1) - \{(a,\tau_{lk}, x)\} & \text{if } \tau_{lk} =HP(FEX^F(t\text{-}1)) \text{ and } x=1 \text{ and } FEX^F(t\text{-}1)\neq\varnothing \text{ and } [t\text{-}1,t] \text{ is a free slot} \\ FEX^F(t\text{-}1) & \text{Otherwise} \end{cases} \quad \ldots \quad \ldots \quad \ldots(VII)$$

The function $FEX^S_{ij}(t)$ selects the highest priority extra copy of tasks in FEX^F_{ij} and reduces the execution time based on the slack available between the finishing time of the current task and the previously completed task. $FEX^S_{ij}(t)$ is defined as follows:

$$FEX^S_{ij}(t)= \begin{cases} FEX^F_{ij} & \text{if s=0} \\ \varnothing & \text{if } FEX^{(s-1)}_{ij}(t)= \varnothing \\ (FEX^{(s-1)}_{ij}(t) - \{(a,\tau_{lk},x)\}) \cup \{(a,\tau_{lk},x\text{-}1)\} & \text{if } \tau_{lk}=HP(FEX_{ij}^{s-1}(t)) \text{ and } x>1 \\ (FEX^{(s-1)}_{ij}(t) - \{(a,\tau_{lk},x)\} & \text{if } \tau_{lk}=HP(FEX_{ij}^{s-1}(t)) \text{ and } x=1 \end{cases} \quad \ldots \quad \ldots \quad \ldots(VIII)$$

Now, for $F=2$ the extra/recovery task instances that are ready to execute at time t for the example task set in Table 1 is determined using equations I-VIII. For $t=0$ to $t=4$ fault occurrence is not detected since two primary copies has not finished execution. So, using the first and last conditions of equation (VII), $FEX^2(0)= FEX^2(1)= FEX^2(2) =FEX^2(3)=\varnothing$. At t=4, using second condition of (VII), $FEX^2(4)=FEX^2_{11}$ since $fin(\tau_{11})=4$. Using the second condition of equation (I), $FEX^2_{11}=\{(fin(\tau_{11}),\tau_{11},F\times C_1)\}=\{(4,\tau_{11},4)\}$. At t=5, using the third condition of (VII), $FEX^2(5)= FEX^S_{11}(5)$ as $fin(\tau_{11})=4 < t < fin(\tau_{21})=6$ and $pre(\tau_{21}) =\tau_{11}$ and $s=slack(fin(\tau_{ij}),t)=0$. Now using the first condition of (VIII), $FEX^0_{11}(5)=FEX^2_{11}$. FEX^2_{11} is known at t=4 and we have $FEX^0_{11}(5)= \{(4, \tau_{11}, 4)\}$. At t=6, using second

condition of VII we have $FEX^2(6)=FEX^2_{21}$ since $fin(\tau_{21})=6$. Following the equation (I), the triplets in FEX^2_{21} is either $Q^1_{21}(2)$ or $Q^2_{21}(2)$. Case 1: All 2 faults have occurred before. So, $Q^1_{21}(2)= SQ^s_{21}(2)= SQ^0_{21}(2)$ where s = slack(fin(pre(ij)),fin(ij)) =slack(4,6)= 0 using (II). $SQ^0_{21}(2)= FEX^f_{pre(\tau ij)}$ since s=0 using the first condition of (III). So, $SQ^0_{21}(2)= FEX^f_{pre(\tau ij)}=FEX^2_{11}=$ $\{(4, \tau_{11}, 4)\}$ and we have, $Q^1_{21}(2)=SQ^0_{21}(2)= \{(4, \tau_{11}, 4)\}$. The total workload at t=6 is $W^1_{21}(2)=4$ using (V). Case 2: One fault occurred before and a new fault has occurred in τ_{21}. Using (IV), $Q^2_{21}(2)= Q^1_{21}(1) \cup \{(fin(\tau_{21}),\tau_{21},2 \times 1)\} = Q^1_{21}(1) \cup \{(6,\tau_{21},2)\}$. Using equation (II) and (III), we have $Q^1_{21}(1)=SQ^0_{21}(1)$since s =slack(fin(pre(τ_{21})),fin(τ_{21})) =slack(fin(τ_{11}),fin(τ_{21}))= slack(4,6) =0. $SQ^0_{21}(1)= FEX^1_{11} =\{(fin(\tau_{11}),\tau_{11}, F\times C_1)\}$ $=\{(4, \tau_{11}, 4)\}$. So, we have, $Q^2_{21}(2)= Q^1_{21}(1) \cup \{(fin(\tau_{21}),\tau_{21}, 2 \times 1)\}= Q^1_{21}(1) \cup \{(6,\tau_{21},2)\}= SQ^0_{21}(1)$ $\cup \{(6,\tau_{21},2)\} =FEX^1_{11} \cup \{(6,\tau_{21},2)\}=\{(4, \tau_{11}, 4), (6,\tau_{21},2)\}$. And the total work load at t=6 is $W^2_{21}(2)=6$. Since $W_{21}{}^2>W_{21}{}^1$ at t=6, $FEX^2_{21}= Q^2_{21}(2) =\{(4, \tau_{11}, 4), (6,\tau_{21},2)\}$ using the fourth condition of (I). At t=6, $FEX^2(6)=\{(4, \tau_{11}, 4), (6,\tau_{21},2)\}$. Other $FEX^2(t)$ are as follows (can be found using equations I-VIII):

$FEX^2(7)=\{(4,\tau_{11},3),(6,\tau_{21},2)\}$ $FEX^2(8)=\{(4, \tau_{11}, 3), (6,\tau_{21},2)\}$
$FEX^2(9)=\{(4,\tau_{11},3),(6,\tau_{21},2)\}$ $FEX^2(10)=\{(4,\tau_{11}, 3), (6,\tau_{21},2)\}$
$FEX^2(11)=\{(4,\tau_{11},3),(11,\tau_{12},4)\}$
$FEX^2(12)=\{(4,\tau_{11},2),(11,\tau_{12},4)\}$
$FEX^2(13)= \{(4,\tau_{11},1), (11,\tau_{12},4)\}$ $FEX^2(14)= \{(11,\tau_{12},4)\}$

In the next subsection 5.3 the set *EX* is defined that combines tasks from *PEX* and *FEX* to find the triplets when tasks are just released and ready to execute.

5.3 Set *EX*: Combining PEX(t) and $FEX^F(t)$

The set EX contains the triplets when a task instance is just released as primary copy or recovery copy in case of worst case distribution of *F* faults. Tasks of the triplets in EX have only $2\times C_i$ or $F\times C_i$ execution time whereas triples in PEX or FEX may have values less than $2\times C_i$ or $F\times C_i$ for primary and extra copies correspondingly. EX is defined as follows:

EX={(t,τ_{ij},X)| (X=$2\times C_i$ and (a, τ_{ij} ,X)$\in$ PEX(t)) or
(X=F×Ci and (a, τ_{ij} ,X) $\in FEX^F(t)$) for t=0,1…PC}

For the task set in Table 1, the set EX for *F*=2 is as follows: EX={(0,τ_{11}, 4), (0,τ_{21}, 2), (4,τ_{11}, 4), (6,τ_{21}, 2), (7,τ_{11}, 4), (11,τ_{11}, 4). It is not the case that all task instances have to run additional *F* copies. If the *F*=1, for the task in Table 1, EX={(0,τ_{11}, 4), (0,τ_{21}, 2), (4,τ_{11}, 2), (7,τ_{12}, 4), (11,τ_{12}, 4)} where no extra copy for τ_{21} needs to run. In next section, the *FT-RT-CMP* algorithm is developed using set *EX* to find the minimum number of cores, denoted by *MinC,* to successfully schedule the task set *EX*.

6. Scheduling Algorithm in CMP:

The *FT-RT-CMP* algorithm, using the ***while*** loop in line 3-44 schedules tasks in set *EX* using *NP* number of cores (line 2). If a task is not schedulable using *NP* cores, *NP* is increased by 1 (line 4). The *MinC* is set at line 7 each time the while loop (line 3-44) starts. For each of the *NP* cores total *PC* free time slots are available and simulated using the 2D array *Slot* at line 10 (initially set to "free"). The ***while*** loop at line 12-41, schedules all the tasks in *EX* using total *NP* cores.

Algorithm: FT-RT-CMP(*Set of Triplets* EX)

```
1  MAX_NP=Maximum number of cores available in a CMP
2  NP=0
3  While (NP < MAX_NP)
4  Label 1: NP=NP+1
5          Set of Triplets EX_Temp=EX
6          //Minimum Number of cores to start the schedule
7          int MinC=NP
8          // For Each of NP cores total PC time slots are
9          //available and set to "free" slot
10         2D-Array of type Task Slot[NP][PC]= {"Free"}
11         //This loop schedules all individual task τij
12         While (Ex_Temp ≠ ∅)
13           Find the highest priority task τlk in EX_Temp
14           For each (a, τlk ,b)∈ EX_Temp
15               Find the number of all copies in TotalCopy
16           Find ReleaseTime and DeadLine for task τlk
17           //All copies of the highest priority task is
18             //scheduled in the following loop
19           Label 2: While (TotalCopy≠0)
20                      SeqTimeUnit=Cl
21             Label 3: For i=ReleaseTime to Deadline
22                        For P=1 to NP
23                          If (Slot[P][i]="Free") then
24                            Slot[P][i]= τlk
25                            SeqTimeUnit= SeqTimeUnit -1
26                            If SeqTimeUnit=0 then
27                              TotalCopy=TotalCopy-1
28                              Goto Label 2
29                            Else
30                               Goto Label 3 for next i
31                            End if
32                          End if
33                        End For
34                      End For
35               If SeqTimeUnit≠0 then
36                  Print "τlk is not schedulable in NP cores"
37                 If NP=MAX_NP then
38                   Print "Task set not schedulable" and Exit
39                     Else Goto Label 1 End if
40             End While
41       End while
42       Print "The task set is Scheduleable"
43       Return Slot and MinC
44 End While
```

Figure 4. Fault tolerant Schedule using CMP

In line 13-15, the highest priority task τ_{lk} is extracted from set *EX* and total number of this task copies that can run in parallel is calculated in variable *TotalCopy*. The release and deadline of τ_{lk} is determined during which task τ_{lk} is scheduled in the ***while*** loop at line 19-40. Note that, each task copy of τ_l should have sequential C_l time unit in the schedule, is stored in *SeqTimeUnit* variable at line 20. The two nested ***For*** loops at line 21-34 check for C_l units of sequential free slots in array *Slot* to schedule the task in single or multiple cores. If *SeqTimeUnit* becomes 0, the task copy is scheduled successfully, *TotalCopy* is decreased by 1 and the ***while*** loop at line 19 iterates for next copy of the same task if available, otherwise, the ***while*** loop at line 12 starts with the next priority task in set *EX*. If *SeqTimeUnit* is not 0 when the nested loop in lines 21-34 is over, the task τ_{lk} can not be scheduled using *NP* number of cores and *NP* is increased by one (line 46 and line 4). If the set *EX* is empty, the task set is schedulable using *MinC* cores and the array *Slot* representing the schedule is returned from line 43. Figure 5 is the real-time fault-tolerant schedule for task in Table 1 for *F*=2. *MinC=2* is determined by the FT-RT-CMP algorithm.

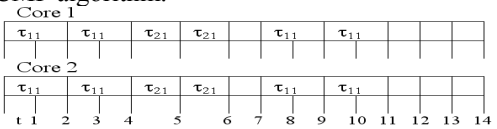


Figure 5. The FT-RT-CMP Schedule for task set in Table 1 using two cores (*MinC*=2)

7. Conclusion

The inherent parallelism within real-time periodic tasks are exploited by CMP in this paper by scheduling the tasks in set *EX* by using the *FT-RT-CMP* algorithm that finds the minimum number of cores *MinC* required for a feasible rate-monotonic fault-tolerant schedule while masking a maximum *F* transient faults. Time redundant execution for fault masking is addressed using CMP's ability to make more CPU time available. More slack become available in the schedule and real-time task set not schedulable in uniprocessor becomes schedulable using CMP even if *F* faults occur. For some task set with high task level parallelism the maximum number of cores can be provided to make the task set schedulable. But providing more cores requires more money and may not be available from CMP industry. In the future, chip microprocessors are expected to support beyond 100 simultaneous threads and can run 100 real-time system tasks in parallel. In future, exact schedulability conditions and other scheduling algorithm like EDF can be considered for chip multiprocessor.

REFERENCES

[1] D. Briere and P. Traverse, "AIRBUS A320/A330/A340 electrical flight controls- A family of fault-tolerant systems", FTCS-23 The Twenty-Third International Symposium on Fault-Tolerant Computing, 22-24 June 1993, Toulouse, France, 1993.

[2] L. Andrade and C. Tenning, "Design of Boeing 777 electric system", IEEE Aerospace and Electronics Systems Magazine, vol. 7, pp 4-11, 1992.

[3] Joakim Aidemark, "Node-Level Fault Tolerance for Embedded Real-Time Systems" Ph. D. Thesis, Chalmers University of Technology, 2004.

[4] Frank Liberato, Rami Melhem, and Daniel Mosse. "Tolerance to multiple faults for aperiodic tasks in hard real time system", IEEE Trans. Computers 49(9):906-914, 2000.

[5] S. Ramos-Thuel. "Enhancing Fault tolerance of Real-time systems through Time redundancy", Ph. D. Thesis, Carnegie Mellon University, May 1993.

[6] L. Hammond, B. Hubbert, M. Siu, M. Chen, K. Olukotum, "The Stanford Hydra CMP", IEEE MICRO Magazine, March-April 2000, and presented at Hot Chips 11, August 1999.

[7] P. Kongetira, K. Aingaran, K. Olukotun, "Niagara: A 32-Way Multithreaded Sparc Processor" , IEEE Micro, Vol 25, No 2, Mar.Apr. 2005, pp 21-19

[8] L. Hammond, B. A. Navfeh, K. Olukotun, "A Single-Chip Multiprocessor", IEEE Computer Special Issue on "Billion-Transistor Processors", September 1997.

[9] D. P. Siewiorek, V. Kini, H. Mashburn, S. McConnel, and M. Tsao, "A Case Study of C.mmp, Cm*, and C.vmp: Part 1: Experiences with Fault Tolerance in Multiprocessor Systems", Proceedings of the IEEE, 66(10):1178-1199, Oct. 1978.

[10] R. K. Iyer, D. J. Rossetti and M. C. Hsueh, "Measurement and Modelling of Computer Reliability as Affected by System Activity", ACM Trans. On Computer Systems, 4(3): 214-237, Aug. 1986.

[11] A. Campbell, P. McDonald, and K. Ray. "Single Event Upset Rates in Space", IEEE Trans. On Nuclear Science, 39(6): 1828-1835, Dec, 1992.

[12] S. Ghosh, R. Mehlem, D. Mosse and J. S. Sarma. "Fault tolerant rate monotonic scheduling", Journal of Real-Time Systems, 15(2), September 1998.

[13] N. Kandasamy, J. P. Hayes, and B.T. Murray, "Tolerating Transient Faults in Statically Scheduled Safety Critical Embedded Systems", Proc. 18th TEEE symposium Reliable Distributed System(SRDS), pp. 212-221, 1999.

[14] A. Burns, R. Davis, and S. Punnekkat, "Feasibility Analysis of Fault Tolerant Real Time Task Sets", In 8th Euromicro Workshop on Real-Time Systems, Jun 1996.

[15] Kumar, S.; Hughes, C.J.; Nguyen, A.; Kumar, A. "Architectural Support for Fine-Grained Parallelism on Multi-core Architectures" Intel Technology Journal. Vol. 11 Issue 03 , August 2007.

[16] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in Hard Real-Time Environment", Journal of ACM, 20(1): 40-61, 1973.

Author Index

Proceedings Companion

EDCC-7